Daftar Isi ……. i
I. Pendahuluan
A. Pengenalan DT51 Debugger ……. 1
B. Istilah pada Instruksi MCS-51
1) Program Status Word (PSW) ……. 2
2) Addressing Modes ……..……. 2
3) Daftar Istilah pada Instruksi MCS-51 …… 3
II. Instruksi Mikrokontroler Keluarga MCS-51
A. Instruksi Aritmetik ……. 4
B. Instruksi Boolean …… 16
C. Instruksi Transfer Data ……. 37
D. Instruksi Percabangan dalam Program ……... 50

I. Pendahuluan

A. Pengenalan DT51 Debugger

DT51 adalah alat pengembangan mikrokontroler keluarga MCS-51™ yang sederhana, handal dan ekonomis. DT51 berbentuk sistem minimum dengan komponen utamanya mikrokontroler AT89C51. DT51 memungkinkan kita bereksperimen sendiri mengembangkan aplikasi digital. DT51 telah dilengkapi dengan debugger DT51D yang akan melacak setiap kesalahan yang ada pada software.

DT51 Debugger, yang selanjutnya disebut DT51D, adalah program debugger/ pencari kesalahan untuk board DT51. Dengan menggunakan DT51D kita dapat dengan cepat dan mudah menemukan *bug*/ kesalahan dalam program kita.

Beberapa kemampuan yang ada pada DT51D, antara lain:

- **Step,** yaitu menjalankan program kita instruksi demi instruksi, di mana setiap kali selesai menjalankan satu instruksi seluruh isi register, flag dapat terlihat pada monitor PC. Untuk menggunakan menu **Step** kita dapat menekan tombol Alt+R | S atau F8.
- **Trace**, hampir sama dengan step namun trace tidak masuk instruksi demi instruksi dalam procedure, sehingga kita dapat melakukan step dengan lebih cepat. Untuk menggunakan menu **Trace** kita dapat menekan tombol Alt+R | T atau F7.
- **Goto Cursor**, yaitu menjalankan program sampai pada posisi kita meletakkan kursor. Untuk menggunakan menu **Goto Cursor** kita dapat menekan tombol Alt+R | G atau F4.
- **Run,** yaitu menjalankan program secara keseluruhan dari DT51D. Untuk menggunakan menu **Run** kita dapat menekan tombol Alt+R | R atau F9.
- **Memory Dump**, di mana kita dapat memonitor isi memori setiap kali satu intruksi dijalankan, bahkan kita dapat menentukan sendiri range memori yang akan dimonitor. Untuk menentukan range memori yang akan dimonitor dapat menekan tombol Alt+M | R. Adapun range memori yang dipakai oleh DT51 adalah 0000h – 007Fh atau 4000h – 5FFFh.
- **Watches,** di mana kita dapat memonitor variabel-variabel penting pada program kita, dimana setiap watch akan ter-*refresh* isinya setiap kali melaksanakan satu instruksi. Untuk menggunakan menu **Watches** kita dapat menekan tombol Alt+W | A kemudian tentukan alamat memori yang akan diwatch.
- **Multiple Breakpoint**, di mana kita dapat menentukan breakpoint di mana saja pada program. Untuk menggunakan menu **Breakpoint** kita dapat menekan tombol Alt+B | A kemudian tentukan alamat memori yang akan di breakpoint.
- **Modify**, dimana kita dapat dengan mudah memodifikasi isi register, flag, memori. Bila kita ingin memodifikasi isi register kita dapat menekan tombol Alt+D | R, tombol Alt+D | F ditekan bila kita ingin memodifikasi isi flag. Jika kita hendak memodifikasi isi memori kita akan menekan tombol Alt+M | M.
- **On-line Help**, yang memudahkan kita dalam menggunakan DT51D. Untuk menjalankan **On-line Help** kita dapat menekan tombol Alt+H | M

Hal-hal yang perlu diperhatikan sebelum memakai DT51D:

1. Pada program assembly kita, stack pointer register (SP) minimum harus 20h. Sebagai contoh: MOV SP,#20h → benar
   MOV SP,#19h → salah
2. Bit addressable 20h.0 dan 20h.1 tidak boleh digunakan dalam source, karena telah digunakan oleh DT51D kernel code.
3. Pada program assembly kita, jangan mengubah nilai register TH1 dan TL1
4. Bit-bit di bawah ini juga jangan diubah nilainya:
   SMOD : Register PCON bit 7
   EA : Register IE bit 7
   ET1 : Register IE bit 3
   PT1 : Register IP bit 3
   TF1 : Register TCON bit 7
   TR1 : Register TCON bit 6
5. High Nibble / Most Significant Nibble (4 bit upper) dari register di bawah ini jangan diubah nilainya:

SCON : 0101XXXX
TMOD : 0010XXXX
Dimana XXXX boleh diubah nilainya.

Apabila ketentuan di atas dilanggar, maka pada saat men-debug program sistem akan hang-up, dan PC harus di-reset ulang. Ketentuan di atas hanya berlaku saat men-debug program dan setelah selesai ketentuan tersebut tidak berlaku lagi. Untuk keterangan yang lebih jelas mengenai pemakaian dan kemampuan dari DT51D dapat dibaca pada DT51D help file pada program DT51D dengan menekan tombol shift+F1, atau melalui menu Help.

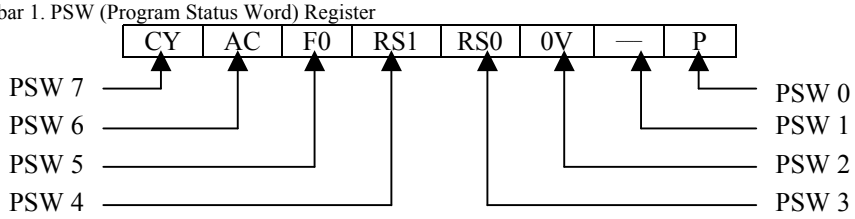
Selain untuk menemukan *bug*/ kesalahan dalam program, DT51D dapat juga membantu kita untuk memahami setiap instruksi yang digunakan pada mikrokontroler keluarga MCS-51.

B. Istilah Pada Instruksi MCS-51

1) Program Status Word (PSW)

Program Status Word (PSW), lihat gambar 1, berisi Carry bit, Auxiliary Carry (untuk BCD), dua register bank, Overflow Flag, Parity bit, dan dua user-definable status Flag.

Gambar 1. PSW (Program Status Word) Register



Keterangan:

PSW 7 : Carry Flag (CY)
PSW 6 : Auxiliary Carry Flag (AC)
PSW 5 : Flag 0 (F0) dapat digunakan oleh user sebagai general purpose flag
PSW 4 : Register Bank selector bit 1 (RS1)
PSW 3 : Register Bank selector bit 0 (RS0)
} RS0 dan RS1 digunakan untuk memilih register bank yang aktif.
PSW 2 : Overflow Flag (OV)
PSW 1 : User definable flag (–)
PSW 0 : Parity Flag (P), merupakan bit even-parity dari Akumulator. Jika banyaknya angka 1 dalam Akumulator ganjil, maka P akan di-set menjadi 1; sebaliknya P akan di-clear menjadi 0.

2) Addressing Modes

Addressing modes yang digunakan pada instruksi MCS-51 adalah:

a. Direct Addressing
   Alamat 8-bit yang menunjukkan lokasi RAM internal (0-127) atau SFR (128-255).
b. Indirect Addressing
   Indirect addressing dapat berisi alamat 8-bit yang terdiri dari Stack Pointer (SP) atau R0 atau R1 dari register bank yang sedang aktif, dan berisi alamat 16-bit yang terdiri dari 16-bit data pointer register (DPTR). Dalam penggunaannya, indirect addressing haruslah diisi dengan alamat data yang dimaksud.
   Contoh: MOV R0,#40h
   MOV 40h,0Fh
   MOV A,@R0
   Instruksi diatas akan menyebabkan register R0 berisi 40h, RAM internal alamat 40h akan berisi data 0Fh dan Akumulator akan berisi 0Fh.
c. Register Instructions
   Instruksi ini akan mengeksekusi register R0-R7 dari register bank yang sedang aktif.

Contoh: MOV A,R7
Setelah instruksi ini dieksekusi maka Akumulator akan berisi register R7.

d. Register-Specific Instructions
Ada beberapa instruksi yang langsung menuju pada register tertentu. Sebagai contoh, beberapa instruksi selalu mengeksekusi Akumulator, jadi tidak ada address byte yang dibutuhkan untuk mengeksekusi instruksi tersebut.
Contoh: DEC A
CPL A

e. Immediate Constants
Nilai konstan yang digunakan dalam instruksi MCS-51. Nilai konstan ini dapat dalam bentuk desimal, heksadesimal maupun biner.
Contoh: MOV A,#100
akan menyebabkan Akumulator berisi bilangan 100 desimal atau bila dinyatakan dalam heksadesimal akan berisi 64h.

f. Indexed Addressing
Program memori hanya dapat diakses oleh indexed addressing. Addressing mode ini dimaksudkan untuk membaca look-up table yang ada di program memori. Indexed addressing ini dipakai pada instruksi JMP @A+DPTR.

3) Daftar Istilah pada Instruksi MCS-51

Ada beberapa istilah yang sering digunakan pada instruksi MCS-51. (lihat tabel 1).

Tabel 1. Daftar Istilah Instruksi MCS-51

Rn	Register R7-R0 dari Register Bank yang sedang aktif.
Direct	Alamat 8-bit yang menunjukkan lokasi RAM internal (0-127) atau SFR (128-255).
@Ri	@R0 atau @R1; digunakan untuk register-indirect addressing terhadap data dalam RAM internal. Sebelum melakukan register-indirect addressing ini, R0 atau R1 harus diisi dengan alamat data yang dimaksud.
#data	Konstanta 8-bit
#data16	konstanta 16-bit
addr16	alamat 16-bit yang digunakan oleh LCALL dan LJMP. Alamat ini bisa berada dalam jangkauan 64 Kbyte (0000h – FFFFh).
addr11	Alamat 11-bit yang digunakan oleh ACALL dan AJMP. Alamat ini harus berada dalam blok 2 Kbyte yang sama dengan perintah berikutnya yang mengikuti ACALL / AJMP tersebut.
rel	offset (pergeseran) 8-bit dalam bentuk bilangan bertanda (two's complement). Digunakan oleh SJMP dan semua conditional jump (JNB, JBC, JC, dsb). Jangkauan alamat ini –128 s/d +127 dari perintah berikutnya yang mengikuti SJMP atau conditional jump tersebut.
bit	Menunjukkan alamat bit dalam RAM internal / SFR yang bersifat bit-addressable.

II. Instruksi Mikrokontroler Keluarga MCS-51

A. Instruksi Aritmetik

Execution time tabel 2 menggunakan frekuensi oscillator sebesar 12 MHz.

Tabel 2. Instruksi Aritmetik

Mnemonic & Operand	Operasi	Execution Time (μs)
ADD A,<byte>	A = A + <byte>	1
ADDC A,<byte>	A = A + <byte> + C	1
SUBB A,<byte>	A = A - <byte> - C	1
INC A	A = A + 1	1
INC <byte>	<byte> = <byte> + 1	1
INC DPTR	DPTR = DPTR + 1	2
DEC A	A = A – 1	1
DEC <byte>	<byte> = <byte> - 1	1
MUL AB	B:A = B x A	4
DIV AB	A = Int [A/B] B = Mod [A/B]	4
DA A	Decimal Adjust	1

ADD A,<src-byte>

Fungsi: Penjumlahan

Deskripsi:

ADD berfungsi untuk menjumlahkan suatu variable <byte> dengan Akumulator, di mana hasil penjumlahan akan berada dalam Akumulator. Jika terdapat carry-out dari bit 7, maka Carry Flag akan di-set menjadi 1; jika tidak maka akan di-clear menjadi 0. Jika terdapat carry-out dari bit 3, maka Auxiliary-carry Flag akan di-set menjadi 1; jika tidak maka akan di-clear menjadi 0. Jika kedua bilangan yang dijumlahkan adalah unsigned integer, maka Carry Flag yang bernilai 1 menandakan terjadinya overflow.

Overflow Flag (OV) akan di-set jika terdapat carry-out dari bit 7 tetapi tidak terdapat carry-out dari bit 6, atau tidak terdapat carry-out dari bit 7 tetapi terdapat carry-out dari bit 6. Selain kondisi tersebut maka OV akan di-clear menjadi 0. Jika kedua bilangan yang dijumlahkan adalah signed-integer, maka nilai 1 pada OV menandakan terjadinya hasil positif pada penjumlahan dua bilangan negatif, atau hasil negatif pada penjumlahan dua bilangan positif.

Terdapat 4 mode addressing yang dapat dipakai di sini : register, direct, register-indirect, dan immediate.

Contoh:

Mula-mula Akumulator berisi 0C3h (1100001lb), dan Register R0 berisi 0AAh (10101010b). Perintah berikut ini: **ADD A,R0** dan akan menghasilkan 6Dh (01101101b) dalam Akumulator, sedangkan AC = 0, C = 1, dan OV = 1.

ADD A,Rn

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) + (Rn)

ADD A,direct

Jumlah byte: 2 Jumlah cycle: 1

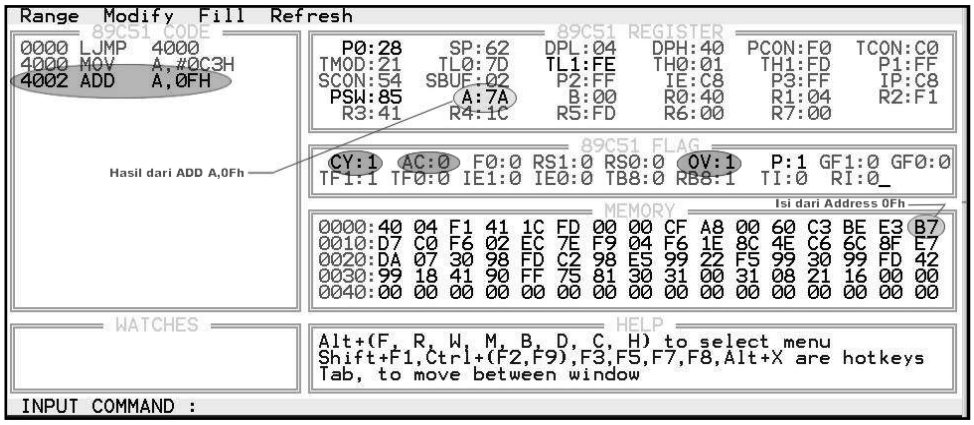
Operasi: (A) ← (A) + (direct)

Contoh:

Mula-mula Akumulator berisi 0C3h (11000011b). Isikan data B7h ke alamat 0Fh dengan memodifikasi isi dari memori alamat 0Fh. Perintah berikut ini: **ADD A,0Fh** akan menghasilkan C3h (11000011b) + B7h (10110111b) = 7Ah (01111010b) dalam Akumulator, sedangkan AC = 0, C = 1, dan OV = 1. Untuk lebih jelasnya lihat gambar2.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 000Fh dan datanya adalah B7h.

Gambar 2. Instruksi ADD A,direct pada DT51D



ADD A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

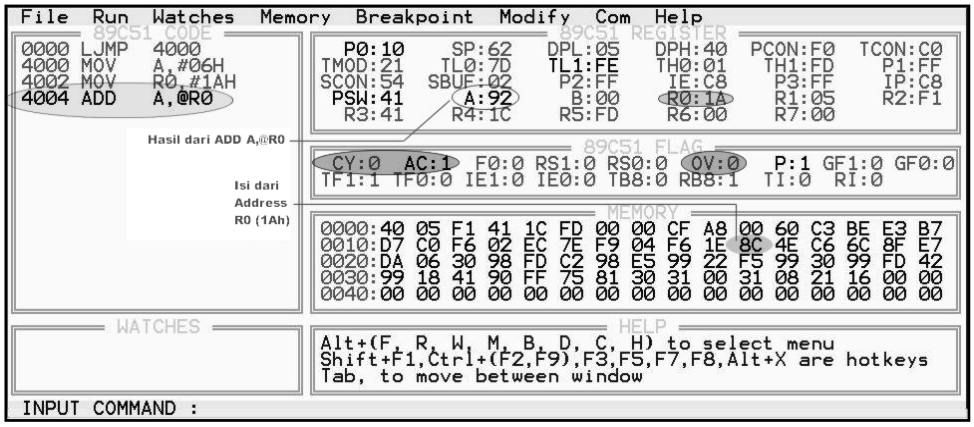
Operasi: (A) ← (A) + ((Ri))

Contoh:

Mula-mula Akumulator berisi 06h (00000110b), dan register R0 berisi 1Ah (00011010b). Isikan data 8Ch ke alamat 1Ah dengan memodifikasi isi dari memori alamat 1Ah. Perintah berikut ini: **ADD A,@R0** dan akan menghasilkan 06h (00000110b) + 8Ch (10001100b) = 92h (10010010b) dalam Akumulator, sedangkan AC = 1, C = 0, dan OV = 0. Lihat gambar 3.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 001Ah dan datanya adalah 8Ch.

Gambar 3. Instruksi ADD A,@Ri pada DT51D



ADD A,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) + #data

ADDC A,<src-byte>

Fungsi: Penjumlahan dengan melibatkan Carry Flag

Deskripsi:

ADDC berfungsi untuk menjumlahkan nilai variabel, Akumulator, dan Carry Flag, dan

meletakkan hasilnya dalam Akumulator. Jika terdapat carry-out dari bit 7, maka Carry Flag akan di-set menjadi 1; jika tidak maka akan di-clear menjadi 0. Jika terdapat carry-out dari bit 3, maka Auxiliary-carry Flag akan di-set menjadi 1; jika tidak maka akan di-clear menjadi 0. Dalam penjumlahan unsigned-integer, nilai 1 pada Carry Flag menandakan terjadinya overflow.
Overflow Flag (OV) akan di-set jika terdapat carry-out dari bit 7 tetapi tidak terdapat carry-out dari bit 6, atau tidak terdapat carry-out dari bit 7 tetapi terdapat carry-out dari bit 6. Selain kondisi tersebut maka OV akan di-clear menjadi 0. Jika kedua bilangan yang dijumlahkan adalah signed-integer, maka nilai 1 pada OV menandakan terjadinya hasil positif pada penjumlahan dua bilangan negatif, atau hasil negatif pada penjumlahan dua bilangan positif.
Terdapat 4 mode addressing yang dapat dipakai di sini : register, direct, register-indirect, dan immediate.

Contoh:

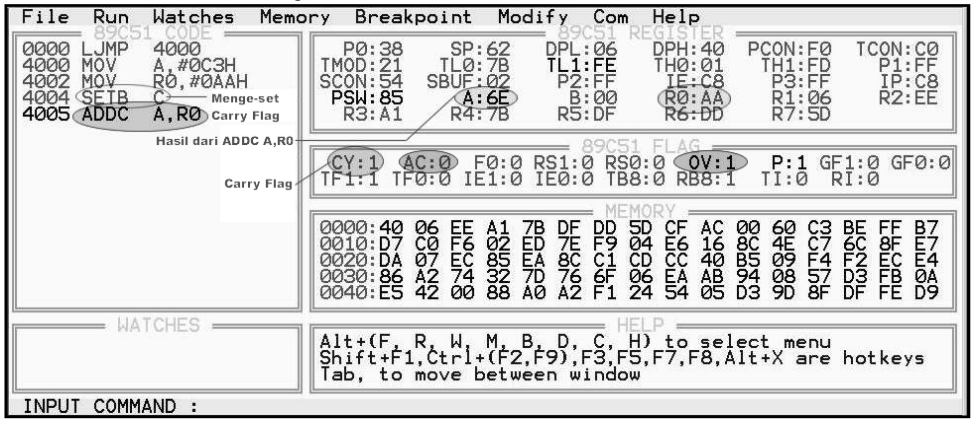
Mula-mula Akumulator berisi 0C3h (11000011b) , Register R0 berisi 0AAh (10101010b), dan Carry Flag bernilai 1. Perintah berikut ini: **ADDC A,R0** akan menghasilkan 6Eh (01101110b) dalam Akumulator, sedangkan AC = 0, C = 1, dan OV = 1. (lihat gambar 4)

ADDC A,Rn

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) + (C) + (Rn)

Gambar 4. Instruksi ADDC A,Rn pada DT51D



ADDC A,direct

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) + (C) + (direct)

ADDC A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) + (C) + ((Ri))

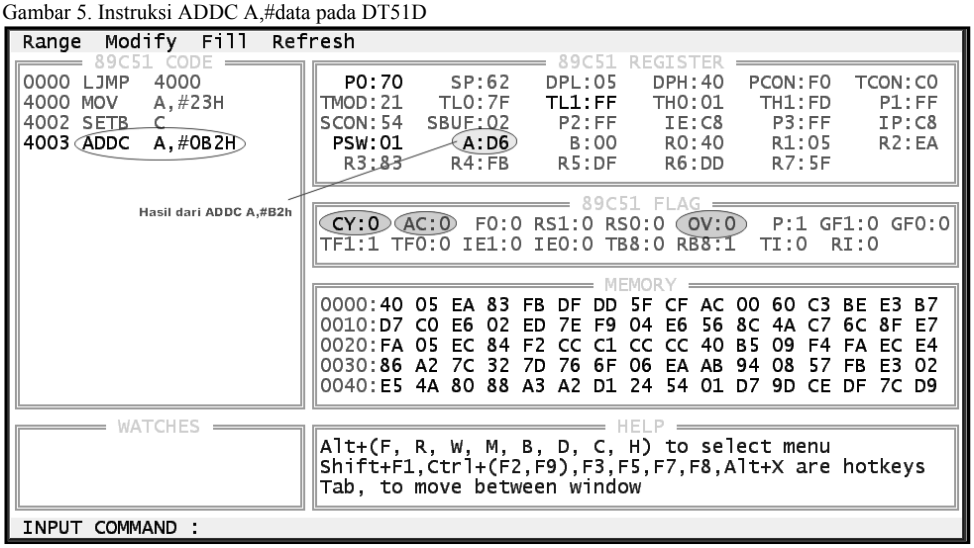
ADDC A,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) + (C) + #data

Mula-mula Akumulator berisi 23h (00100011b). Perintah berikut ini: **ADDC A,#0B2h** akan menghasilkan 23h (00100011b) + 1h (00000001b) + B2h (10110010b) = 0D6h (11010110b) dalam Akumulator, sedangkan AC = 0, C = 0, OV = 0. (lihat gambar 5)

Gambar 5. Instruksi ADDC A,#data pada DT51D



SUBB A,<src-byte>

Fungsi: Pengurangan dengan melibatkan Carry Flag

Deskripsi:

SUBB berfungsi untuk mengurangi Akumulator dengan suatu nilai dalam src-byte dan Carry Flag, dan meletakkan hasilnya dalam Akumulator (Acc ← Acc ← [src-byte] - C). Jika diperlukan borrow pada bit 7, maka Carry Flag akan di-set menjadi 1; jika tidak maka Carry Flag akan di-clear menjadi 0. Selanjutnya Carry Flag ini dapat digunakan pada perintah SUBB berikutnya, sehingga terjadi pengurangan multiple precision.

Jika diperlukan borrow pada bit 3, maka AC akan di-set menjadi 1; jika tidak maka AC akan di-clear menjadi 0. Sedangkan Overflow Flag (OV) akan di-set jika diperlukan borrow untuk bit-7 tetapi tidak diperlukan borrow untuk bit-6, atau jika tidak diperlukan borrow untuk bit-7 tetapi diperlukan borrow untuk bit-6.

Pada pengurangan signed integer, nilai 1 pada OV menandakan terjadinya hasil negatif pada pengurangan bilangan positif dengan bilangan negatif, atau terjadinya bilangan positif pada pengurangan bilangan negatif dengan bilangan positif.

Terdapat 4 mode addressing yang dapat digunakan : register, direct, register-indirect, atau immediate.

Contoh:

Mula-mula Akumulator bernilai 0C9h (11001001b), Register R2 bernilai 54h (01010100b), dan Carry Flag bernilai 1. Perintah berikut ini: **SUBB A,R2** akan menghasilkan 74h (01110100b) pada Akumulator, sedangkan C = 0, AC = 0, dan OV = 1. Perhatikan bahwa dalam perintah SUBB, nilai Carry Flag selalu ikut dikurangkan dari Akumulator. Karena itu jika pada saat perintah SUBB dieksekusi kondisi Carry Flag tidak diketahui, maka harus ditambahkan perintah CLR C sebelum perintah SUBB tersebut.

SUBB A,Rn

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) - (C) - (Rn)

SUBB A,direct

Jumlah byte: 2 Jumlah cycle: 1

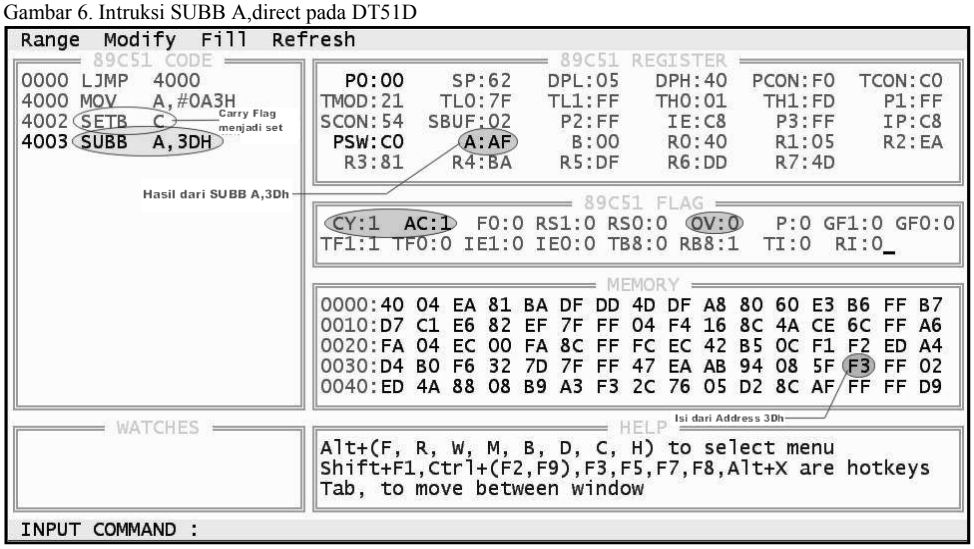
Operasi: (A) ← (A) - (C) - (direct)

Contoh:

Mula-mula Akumulator berisi 0A3h (10100011b) dan Carry Flag bernilai 1. Isikan data F3h ke alamat 3Dh dengan memodifikasi isi dari memori alamat 3Dh. Perintah berikut ini: **SUBB A,3Dh** akan menghasilkan AFh (10101111b) pada Akumulator, sedangkan C = 1, AC = 1, OV = 0. (lihat gambar 6)

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 003Dh dan datanya adalah F3h.

Gambar 6. Intruksi SUBB A,direct pada DT51D



SUBB A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

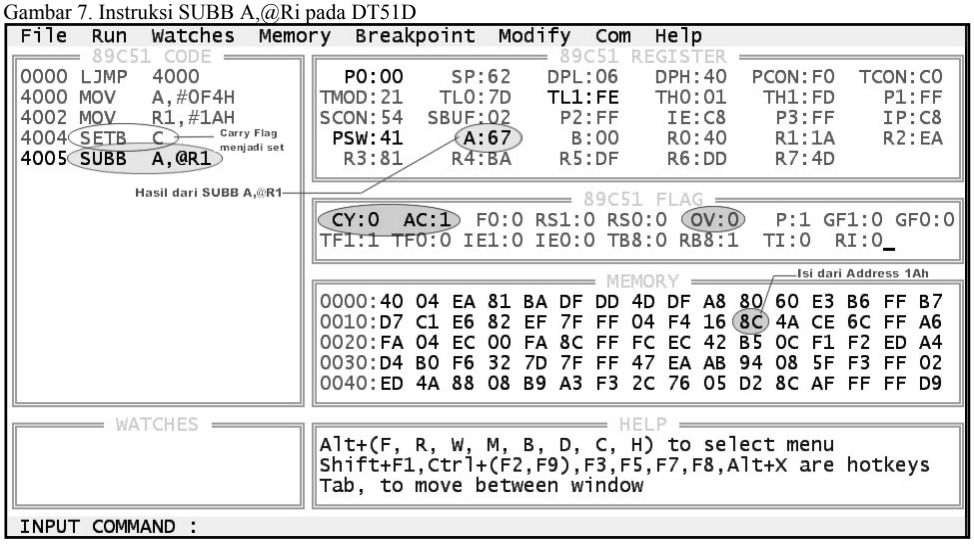
Operasi: (A) ← (A) - (C) - ((Ri))

Contoh:

Mula-mula Akumulator berisi 0F4h (11110100b), Register R1 berisi 1Ah (00011010b) dan Carry Flag bernilai 1. Isikan data 8Ch ke alamat 1Ah dengan memodifikasi isi dari memori alamat 1Ah. Perintah berikut ini: **SUBB A,@R1** akan menghasilkan 0F4h (11110100b) - 1h (00000001b) - 8Ch (10001100b) = 67h (01100111b) pada Akumulator, sedangkan C = 0, AC= 1, OV = 0. Lihat gambar 7

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 001Ah dan datanya adalah 8Ch.

Gambar 7. Instruksi SUBB A,@Ri pada DT51D



SUBB A,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) - (C) - #data

MUL AB

Fungsi: Perkalian

Deskripsi:

MUL AB berfungsi untuk mengalikan bilangan unsigned-integer di Akumulator dan register B. Hasil perkalian akan berukuran 16-bit, di mana low-byte akan berada di Akumulator dan high-byte berada di B. Jika hasil kali lebih dari 255 (0FFh), maka Overflow Flag akan di-set menjadi 1; jika tidak maka akan di-clear menjadi 0. Sedangkan Carry Flag akan selalu di-clear menjadi 0.

Jumlah byte: 2 Jumlah cycle: 1

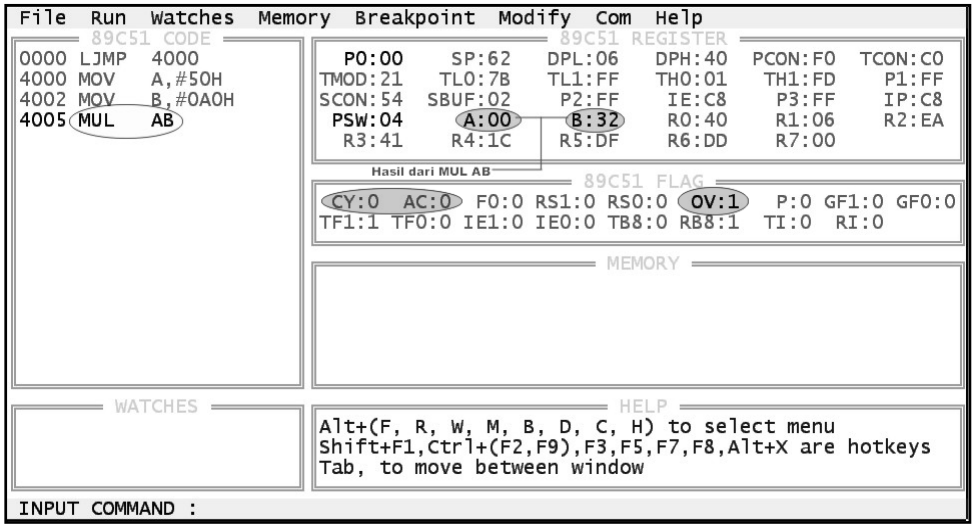
Operasi: $(A)_{7\text{-}0} \leftarrow (A) \times (B)$

$(B)_{15\text{-}8}$

Contoh:

Mula-mula Akumulator berisi bilangan 80 (50h), dan Register B berisi 160 (0A0h). Perintah berikut ini: **MUL AB** akan menghasilkan hasil kali 12800 (3200h), sehingga B akan berisi 32h (00110010b) dan Akumulator berisi 0, sedangkan OV = 1 dan C = 0. Lihat gambar 8.

Gambar 8. Instruksi MUL AB pada DT51D



DIV AB

Fungsi: Pembagian

Deskripsi:

DIV AB berfungsi untuk membagi bilangan dalam Akumulator dengan bilangan dalam register B. Kedua bilangan tersebut dianggap sebagai bilangan unsigned-integer. Hasil bagi (quotient) akan terletak dalam Akumulator, sedangkan sisa pembagian (remainder) terletak dalam register B. Baik Carry Flag maupun Overflow Flag akan di-clear "0."

Perkecualian: Jika B (pembagi) mula-mula berisi 00H, maka hasil bagi dalam Akumulator dan sisa pembagian dalam register B tidak terdefinisikan, dan OV akan di-set menjadi 1. Sedangkan Carry Flag tetap di-clear menjadi 0.

Jumlah byte: 1 Jumlah cycle: 4

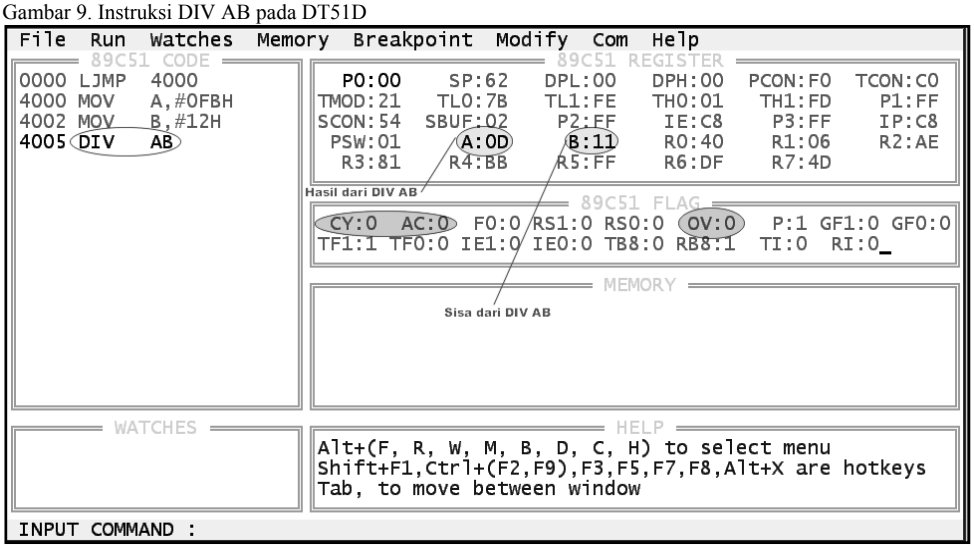
Operasi: $(A)_{15\text{-}8} \leftarrow (A) / (B)$

$(B)_{7\text{-}0}$

Contoh:

Mula-mula Akumulator berisi 251 (0FBh atau 11111011b) dan Register B berisi 18 (12h atau 00010010b). Perintah berikut ini: **DIV AB** akan menghasilkan bilangan 13 dalam Akumulator (0Dh or 00001101b) dan 17 (11h atau 00010001b) dalam B, karena 251 = (13 x 18) + 17. Baik Carry Flag maupun OV di-clear menjadi 0. Lihat gambar 9.

Gambar 9. Instruksi DIV AB pada DT51D



DEC byte

Fungsi: Decrement

Deskripsi:

DEC berfungsi untuk mengurangi suatu variabel byte dengan 1. Jika bilangan yang di-decrement mula-mula adalah 00H, maka akan menghasilkan bilangan 0FFH (underflow). Tidak ada flag yang terpengaruh oleh perintah ini (sekalipun telah terjadi underflow !).

Terdapat 4 mode addressing yang dapat diterapkan di sini : Akumulator, register, direct, dan register-indirect.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

Mula-mula R0 berisi 7Fh (01111111b), sedangkan RAM internal alamat 7Eh and 7Fh berturut-turut berisi data 00h and 40h. Perintah berikut ini :

DEC @R0
DEC R0
DEC @R0

menyebabkan R0 berisi 7Eh, sedangkan RAM internal alamat 7Eh and 7Fh akan berisi 0FFh and 3Fh.

DEC A

Jumlah byte: 1 Jumlah cycle: 1
Operasi: (A) ← (A) - 1

DEC Rn

Jumlah byte: 1 Jumlah cycle: 1
Operasi: (Rn) ← (Rn) - 1

DEC direct

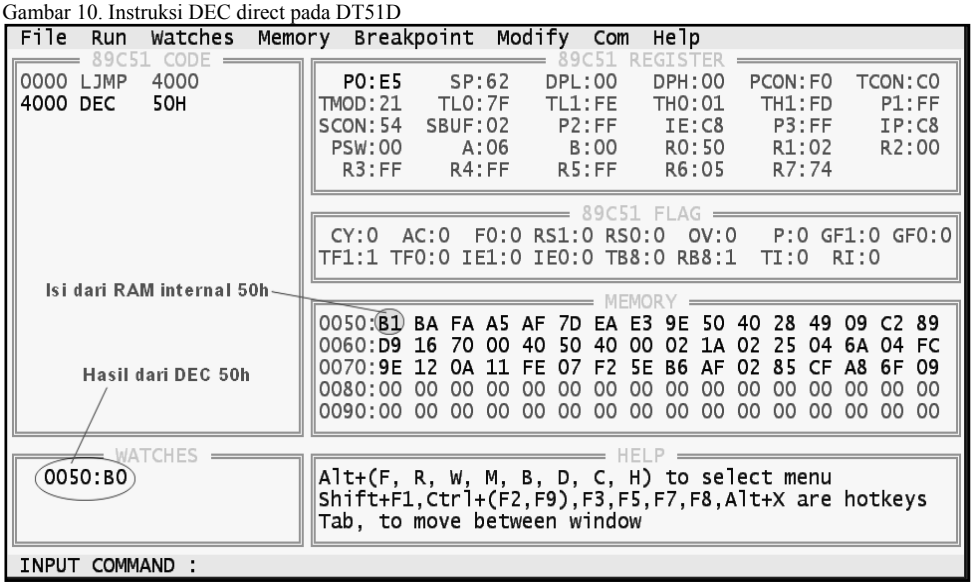
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (direct) ← (direct) – 1
Contoh:

Mula-mula diisikan data B1h ke RAM internal alamat 50h dengan memodifikasi isi dari memori alamat 50h. Perintah berikut ini: **DEC 50h** menyebabkan RAM internal dari alamat 50h akan berisi B0h. Lihat gambar 10.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0050h dan datanya adalah B1h.

Untuk memonitor isi dari RAM internal alamat 50h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0050h.

Gambar 10. Instruksi DEC direct pada DT51D



DEC @Ri

Jumlah byte: 1 Jumlah cycle: 1

Operasi: ((Ri)) ← ((Ri)) – 1

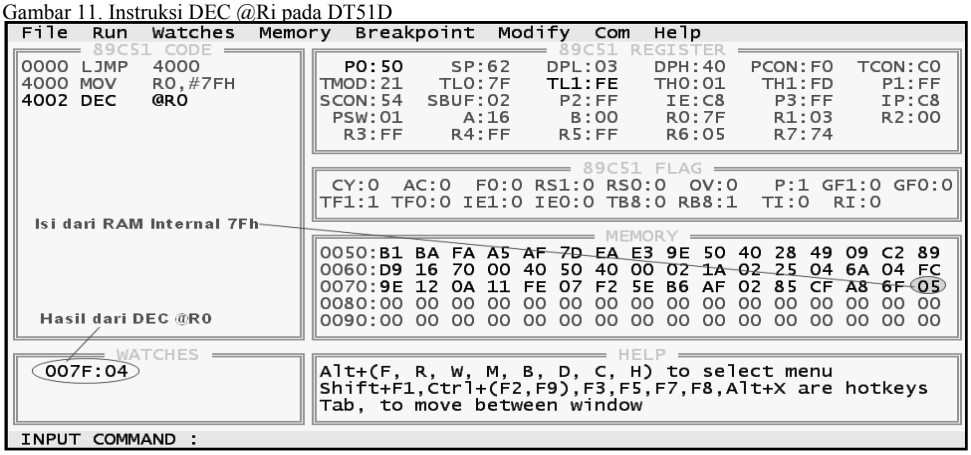
Contoh:

Mula-mula Register R0 berisi 7Fh (01111111b), dan data 05h diisikan ke RAM internal alamat 7Fh dengan memodifikasi isi dari memori alamat 7Fh. Perintah berikut ini: **DEC @R0** menyebabkan RAM internal alamat 7Fh akan berisi 04h. Lihat Gambar 11.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 007Fh dan datanya adalah 05h.

Untuk memonitor isi dari RAM internal alamat 7Fh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 007Fh.

Gambar 11. Instruksi DEC @Ri pada DT51D



INC byte

Fungsi: Increment

Deskripsi:

INC berfungsi untuk meng-increment suatu variabel byte. Variabel byte bernilai 0FFh akan overflow menjadi 00h. Tidak ada flag yang terpengaruh oleh perintah ini.

Terdapat 3 mode addressing yang dapat digunakan : register, direct, dan register-indirect.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

R0 mula-mula berisi 7Eh (011111110b), sedangkan RAM internal alamat 7Eh and 7Fh berturut-turut berisi 0FFh dan 40h. Perintah berikut ini :

INC @R0
INC R0
INC @R0

menyebabkan R0 berisi 7Fh dan RAM internal alamat 7Eh and 7Fh berturut-turut berisi 00h dan 41h.

INC A

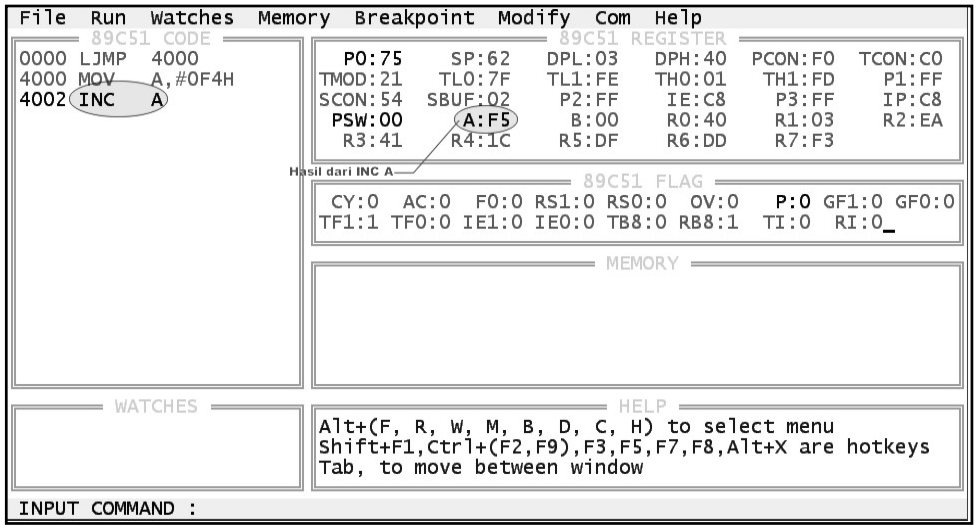
Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) + 1

Contoh:

Akumulator berisi F4h (11110100b), kemudian diberi instruksi **INC A** maka isi dari Akumulator adalah F4h (11110100b) + 1h (00000001b) = F5h (11110101b). Lihat gambar 12.

Gambar 12. Instruksi INC A pada DT51D



INC Rn

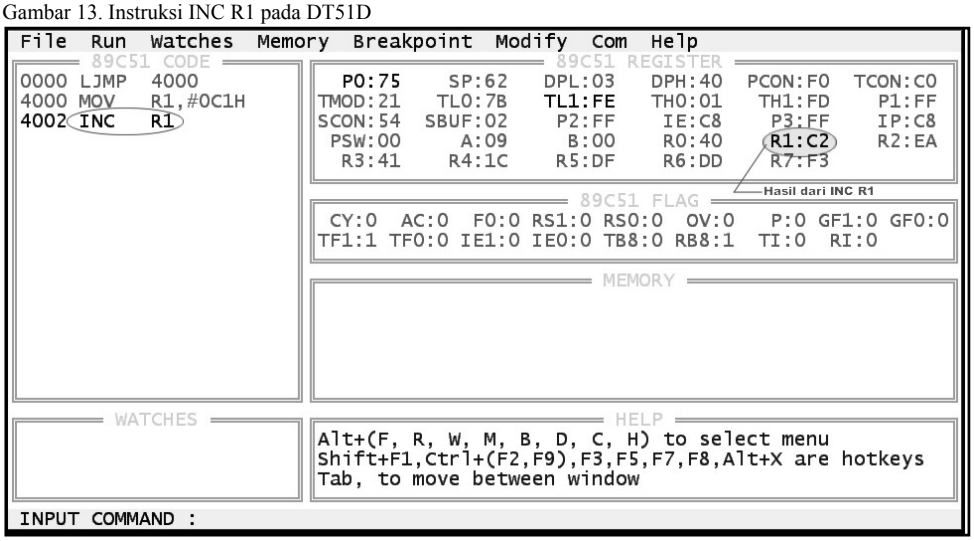
Jumlah byte: 1 Jumlah cycle: 1

Operasi: (Rn) ← (Rn) + 1

Contoh:

Register R1 berisi C1h (11000001b), kemudian diberi instruksi **INC R1** maka isi dari Register R1 adalah C1h (11000001b) + 1h (00000001b) = C2h (11000010b). Lihat gambar 13.

Gambar 13. Instruksi INC R1 pada DT51D



INC direct
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (direct) ← (direct) + 1

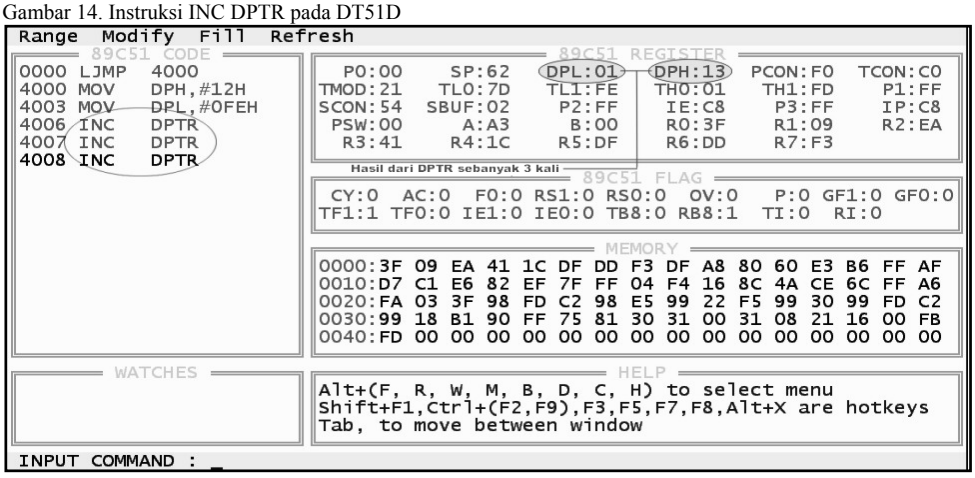
INC @Ri
Jumlah byte: 1 Jumlah cycle: 1
Operasi: (Ri) ← (Ri) + 1

INC DPTR

Fungsi: Increment DPTR (Data Pointer)
Deskripsi:
INC DPTR berfungsi untuk meng-increment DPTR sebagai bilangan 16-bit. Tidak ada flag yang terpengaruh oleh perintah ini.
DPTR adalah satu-satunya register 16-bit yang dapat di-increment.
Contoh:
Register DPH berisi 12h dan Register DPL berisi FEh, kemudian diberi instruksi **INC DPTR** sebanyak 3 kali. Hal ini dilakukan agar register DPH dapat diincrementkan. Hasil dari INC DPTR itu adalah Register DPH menjadi 13h, sedangkan Register DPL menjadi 01h. Lihat gambar 14

Jumlah byte: 1 Jumlah cycle: 2
Operasi: (DPTR) ← (DPTR) + 1

Gambar 14. Instruksi INC DPTR pada DT51D



DA A

Fungsi: ”Decimal-adjust” Akumulator setelah penjumlahan.

Deskripsi:

DA A digunakan setelah penjumlahan bilangan BCD. Perintah ini akan mengubah bilangan biner 8-bit dalam Akumulator (hasil penjumlahan sebelumnya) menjadi dua buah bilangan 4-bit (BCD). DA A dapat digunakan setelah perintah ADD atau ADDC.

Jika bit 3-0 Akumulator bernilai lebih dari 9 (xxxx1010-xxxx1111), atau jika AC = 1, maka nilai Akumulator akan ditambah dengan 6 sehingga menghasilkan 1 digit bilangan BCD pada posisi low-nibble Akumulator (A.3-A.0).

Setelah itu, jika bit 7-4 Akumulator bernilai lebih dari 9 (1010xxxx-1111xxxx), atau jika C = 1, maka nilai Akumulator high-nibble (A.7-A.4) akan ditambah dengan 6, sehingga menghasilkan 1 digit bilangan BCD pada posisi high-nibble Akumulator ini.

Jika setelah semua proses di atas dilakukan ternyata nilai Carry Flag = 1, berarti dapat disimpulkan bahwa penjumlahan kedua bilangan BCD pada perintah ADD/ADDC sebelumnya menghasilkan bilangan lebih dari 100. Carry Flag ini dapat diikutkan pada perintah ADDC selanjutnya sehingga dapat dilakukan penjumlahan bilangan BCD multiple precision.

Overflow Flag (OV) tidak terpengaruh oleh perintah ini.

Catatan: DA A tidak dapat mengkonversi bilangan heksadesimal dalam Akumulator menjadi bilangan BCD. DA A juga tidak dapat diterapkan pada pengurangan bilangan desimal.

Contoh:

Akumulator mula-mula bernilai 56h (01010110b), yang dapat dibaca sebagai bilangan desimal 56 yang ditulis dalam format BCD. Register R3 bernilai 67h (01100111B) yang dapat dibaca sebagai bilangan desimal 67 yang ditulis dalam format BCD. Carry Flag mula-mula bernilai 1. Perintah-perintah berikut ini :

```
ADDC A,R3
DA A
```

akan menjumlahkan A, R3, dan C dan menghasilkan 0BEh (10111110b) dalam Akumulator, dengan C = 0 dan AC = 0.

Selanjutnya DA A mengubah nilai Akumulator menjadi 24h (00100100b), yang menyatakan bilangan desimal 24 yang ditulis dalam format BCD. Selain itu DA A juga meng-set Carry Flag menjadi 1, yang dapat diartikan bahwa penjumlahan ADDC sebelumnya menghasilkan bilangan lebih dari 100, sehingga hasil akhirnya dapat dibaca sebagai 124. Jadi, dapat disimpulkan bahwa 56 + 67 + 1 = 124. Lihat gambar 15

Jumlah byte: 1 Jumlah cycle: 1

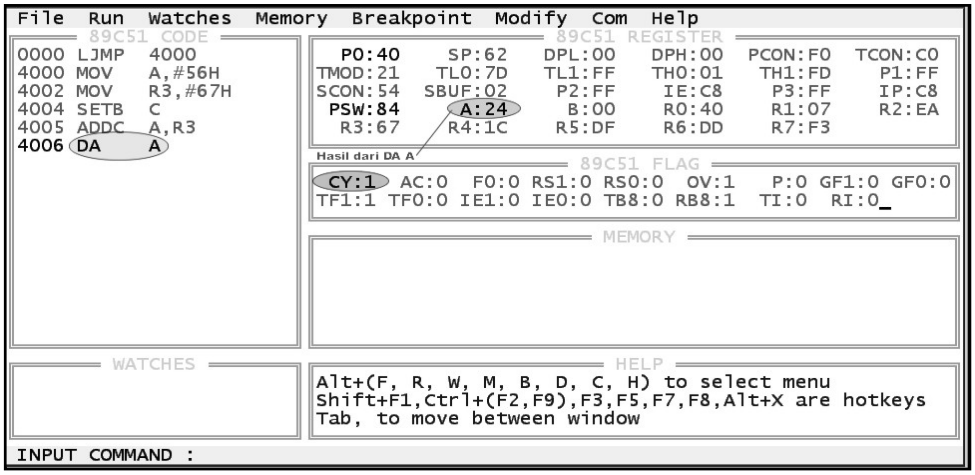
Operasi: jika $[[(A_{3-0}) > 9] \text{ V } [(AC) = 1]]$

maka $(A_{3-0}) \leftarrow (A_{3-0}) + 6$

jika $[[(A_{7-4}) > 9] \text{ V } [(C) = 1]]$

maka $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

Gambar 15. Instruksi DA A pada DT51D



B. BOOLEAN INSTRUCTIONS

Macam-macam instruksi Boolean dapat dilihat di tabel 3.

Tabel 3. Instruksi Boolean

Mnemonic	Operasi	Execution Time (μs)
ANL A,<byte>	A = A .AND. <byte>	1
ANL C,bit	C = C .AND. bit	2
ANL C./bit	C = C .AND. .NOT. bit	2
ANL <byte>,A	<byte> = <byte> .AND. A	1
ANL <byte>,#data	<byte> = <byte> .AND. #data	2
ORL A,<byte>	A = A .OR. <byte>	1
ORL C,bit	C = C .OR. bit	2
ORL C,/bit	C = C .OR. .NOT. bit	2
ORL <byte>,A	<byte> = <byte> .OR. A	1
ORL <byte>,#data	<byte> = <byte> .OR. #data	2
XRL A,<byte>	A = A .XOR. <byte>	1
XRL <byte>,A	<byte> = <byte> .XOR. A	1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	2
CLR A	A = 00h	1
CLR C	C = 0	1
CLR bit	bit = 0	1
CPL A	A = .NOT. A	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
RL A	Menggeser ACC ke kiri 1 bit	1
RLC A	Menggeser ke kiri lewat Carry	1
RR A	Menggeser ACC ke kanan 1 bit	1
RRC A	Menggeser ke kanan lewat C	1
SWAP A	Swap nibbles di A	1
SETB C	C = 1	1
SETB bit	bit = 1	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1; CLR bit	2

ANL <dest-byte>,<src-byte>

Fungsi: Meng-AND-kan kedua variabel (dest-byte dan src-byte)

Deskripsi:

ANL berfungsi untuk meng-AND-kan (secara bitwise) kedua variabel operand, dan meletakkan hasilnya dalam variabel tujuan (dest-byte). Tidak ada flag yang dipengaruhi perintah ini.

Terdapat 6 kombinasi yang dapat diterapkan pada kedua operand. Jika dest-byte adalah Akumulator, maka src-byte dapat berupa register, direct, register-indirect, atau immediate addressing. Jika dest-byte adalah direct address, maka src-byte dapat berupa Akumulator atau immediate data.

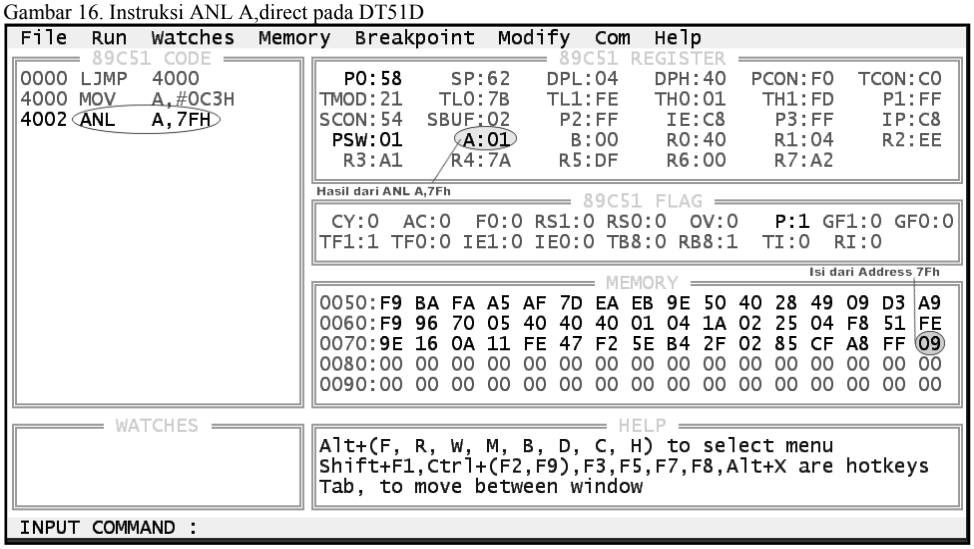
Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:
Mula-mula Akumulator berisi 0C3h (11000011b) dan R0 berisi 55h (01010101b). Perintah berikut ini :
ANL A,R0
akan menghasilkan 41h (01000001b) dalam Akumulator.

ANL A,Rn
Jumlah byte: 1 Jumlah cycle: 1
Operasi: (A) ← (A) Λ (Rn)

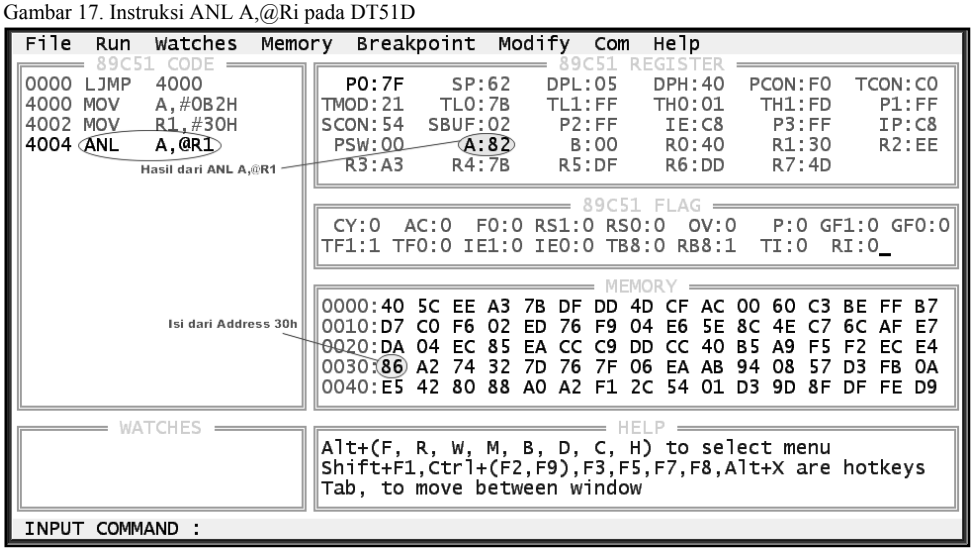
ANL A,direct
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (A) ← (A) Λ (direct)
Contoh:
Mula-mula Akumulator berisi 0C3h (11000011b), data 09h (00001001b) diisikan ke RAM internal alamat 7Fh dengan memodifikasi memori alamat 7Fh. Instruksi berikut:
ANL A,7Fh menyebabkan Akumulator berisi 01h (00000001b). Lihat gambar 16.
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 007Fh dan datanya adalah 09h.

Gambar 16. Instruksi ANL A,direct pada DT51D



ANL A,@Ri
Jumlah byte: 1 Jumlah cycle 1
Operasi: (A) ← (A) Λ ((Ri))
Contoh:
Mula-mula Akumulator berisi 0B2h (10110010b), Register R1 berisi 30h (00110000b), data 86h diisikan ke RAM internal alamat 30h dengan memodifikasi isi dari memori alamat 30h. Perintah berikut:
ANL A,@R1 menyebabkan Akumulator berisi 82h (10000010b). Lihat gambar 17
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0030h dan datanya adalah 86h.

Gambar 17. Instruksi ANL A,@Ri pada DT51D

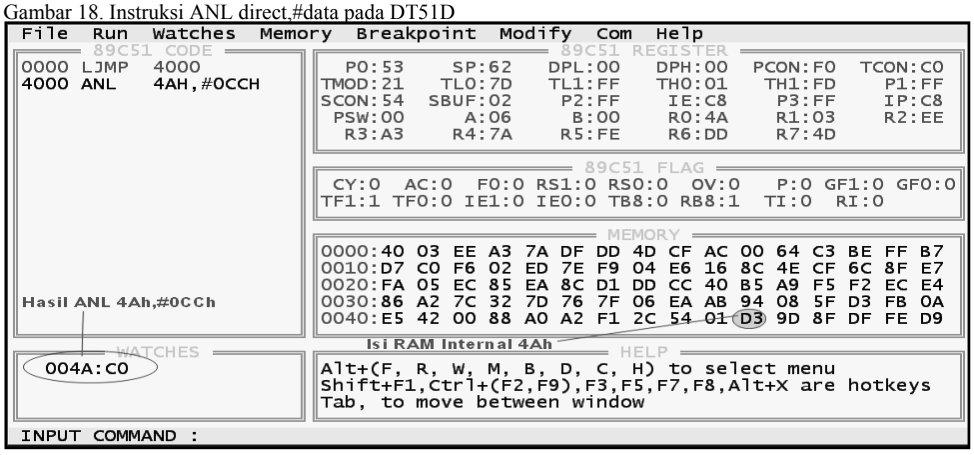


ANL A,#data
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (A) ← (A) Λ #data

ANL direct,A
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (A) ← (direct) Λ (A)

ANL direct,#data
Jumlah byte: 3 Jumlah cycle: 2
Operasi: (A) ← (direct) Λ #data
Contoh:
Mula-mula data 0D3h (11010011b) diisikan ke RAM internal alamat 4Ah dengan memodifikasi isi dari RAM internal 4Ah. Perintah berikut ini:
ANL 4Ah,#0CCh menyebabkan RAM internal alamat 4Ah berisi data 0C0h (11000000b). Untuk memonitor isi dari RAM internal alamat 4Ah digunakan fasilitas Watches yang tersedia pada DT51D. Lihat gambar 18.
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 004Ah dan datanya adalah D3h.
Untuk memonitor isi dari RAM internal alamat 4Ah digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 004Ah.

Gambar 18. Instruksi ANL direct,#data pada DT51D



ORL <dest-byte>,<src-byte>

Fungsi: Meng-OR-kan kedua variabel (dest-byte dan src-byte)

Deskripsi:

ORL berfungsi untuk meng-OR-kan dest-byte dengan src-byte, dan meletakkan hasilnya di dest-byte. Tidak ada flag yang terpengaruh oleh perintah ini.

Terdapat 6 kombinasi addressing yang dapat diterapkan pada kedua operand. Jika dest-byte berupa Akumulator, maka src-byte dapat berupa register, direct, register-indirect, atau immediate addressing. Jika dest-byte berupa direct address, maka src-byte dapat berupa Akumulator atau immediate data.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

Mula-mula Akumulator berisi 0C3h (11000011b) dan R0 berisi 55h (01010101b). Perintah berikut ini :

ORL A,R0

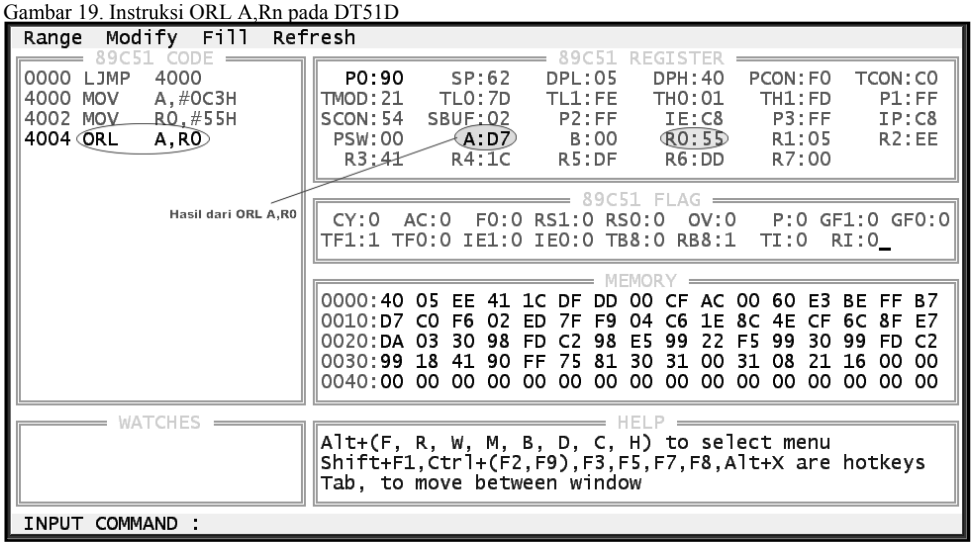
menyebabkan Akumulator bernilai 0D7h (1101011lb). Lihat gambar 19

ORL A,Rn

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) V (Rn)

Gambar 19. Instruksi ORL A,Rn pada DT51D



ORL A,direct

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) V (direct)

ORL A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) V ((Ri))

ORL A,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) V #data

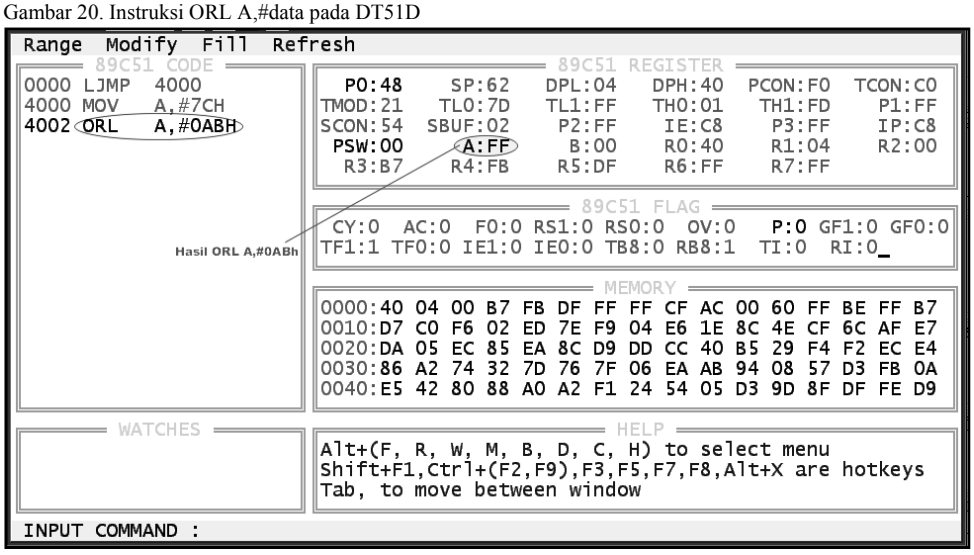
Contoh:

Mula-mula Akumulator berisi 7Ch (01111100b). Perintah berikut:

ORL A,#0ABh

menyebabkan Akumulator berisi 0FFh (11111111b). Lihat gambar 20

Gambar 20. Instruksi ORL A,#data pada DT51D



ORL direct,A

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (direct) V (A)

Contoh:

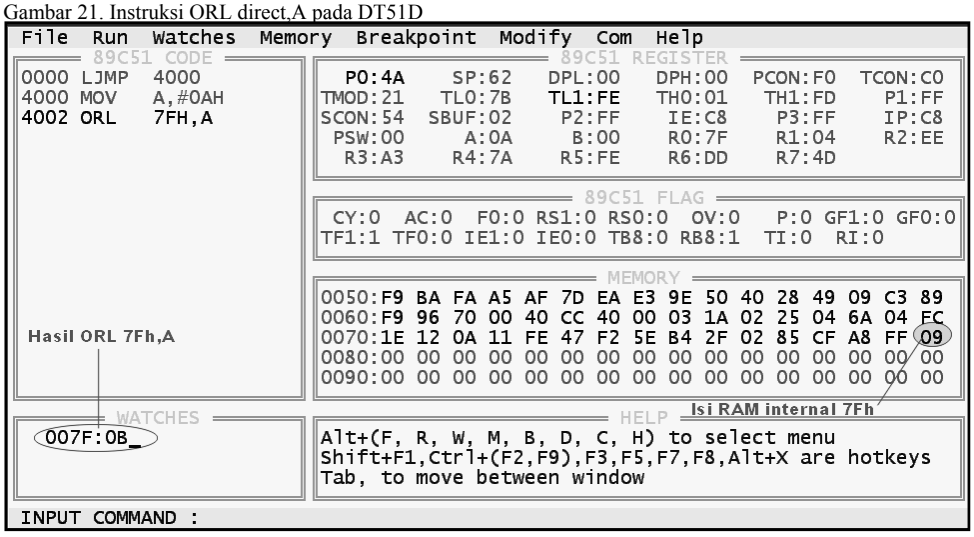
Mula-mula data 09h (00001001b) diisikan ke RAM internal alamat 7Fh dengan memodifikasi isi dari RAM internal 7Fh, Akumulator berisi 0Ah (00001010b). Perintah berikut:

ORL 7Fh,A menyebabkan RAM internal alamat 7Fh berisi data 0Bh (00001011b). Untuk memonitor isi RAM Internal alamat 7Fh digunakan fasilitas Watches yang tersedia pada DT51D. Lihat gambar 21

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 007Fh dan datanya adalah 09h.

Untuk memonitor isi dari RAM internal alamat 7Fh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 007Fh.

Gambar 21. Instruksi ORL direct,A pada DT51D



ORL direct,#data

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (A) ← (direct) V #data

XRL <dest-byte>,<src-byte>

Fungsi: Meng-XOR-kan kedua variabel (dest-byte dan src-byte)

Deksripsi:

XRL berfungsi untuk meng-XOR-kan dest-byte dengan src-byte, dan meletakkan hasilnya di dest-byte. Tidak ada flag yang terpengaruh oleh perintah ini.

Terdapat 6 kombinasi addressing yang dapat diterapkan pada kedua operand. Jika dest-byte berupa Akumulator, maka src-byte dapat berupa register, direct, register-indirect, atau immediate addressing. Jika dest-byte berupa direct address, maka src-byte dapat berupa Akumulator atau immediate data.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

Mula-mula Akumulator berisi 0C3h (1100001lb) dan R0 berisi 0AAh (10101010b). Perintah berikut ini :

XRL A,R0

akan menyebabkan Akumulator berisi 69h (01101001b).

XRL A,Rn

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) ⊻ (Rn)

XRL A,direct

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (A) ⊻ (direct)

Contoh:

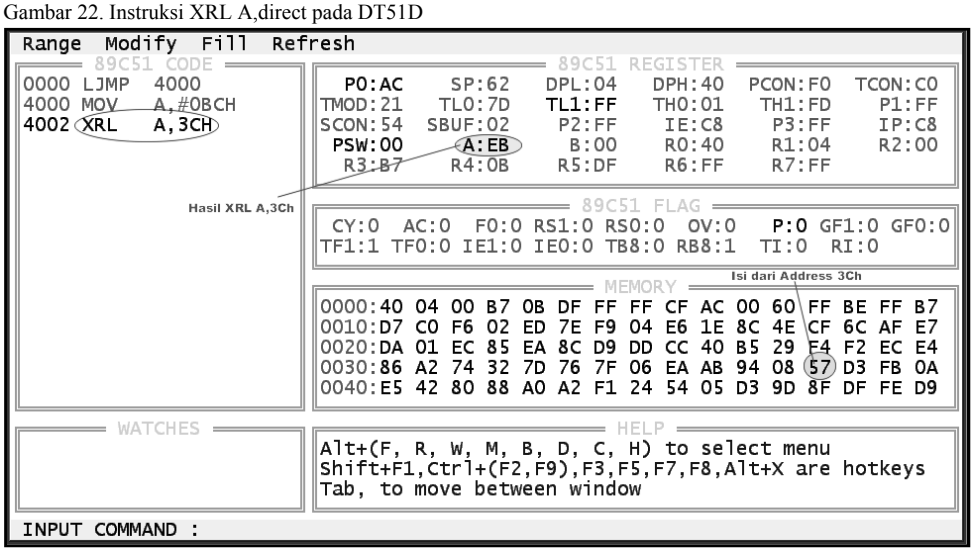
Mula-mula Akumulator berisi 0BCh (10111100b), data 57h diisikan ke RAM internal alamat 3Ch dengan memodifikasi isi dari memori alamat 3Ch. Perintah berikut ini:

XRL A,3Ch

menyebabkan Akumulator berisi 0EBh (11101011b). Lihat gambar 22.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 003Ch dan datanya adalah 57h.

Gambar 22. Instruksi XRL A,direct pada DT51D



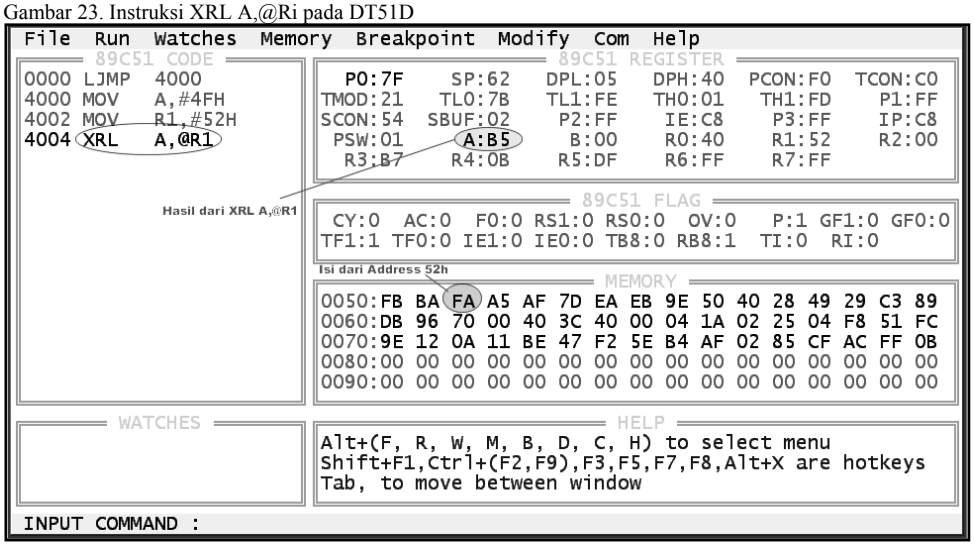
XRL A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A) ← (A) ⊻ ((Ri))

Contoh:
Mula-mula Akumulator berisi 4Fh (01001111b), Register R1 berisi 52h (01010010b), data 0FAh (11111010b) diisikan ke RAM internal alamat 52h dengan memodifikasi memori alamat 52h. Perintah berikut:
XRL A,@R1
menyebabkan Akumulator berisi0B5h (10110101b). Lihat gambar 23
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0052h dan datanya adalah FAh.

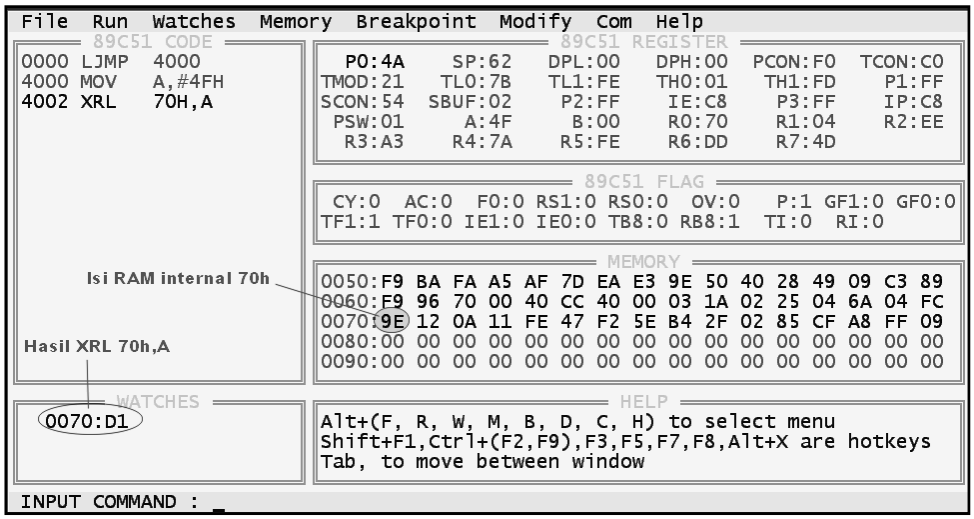
Gambar 23. Instruksi XRL A,@Ri pada DT51D



XRL A,#data
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (A) ← (A) V #data

XRL direct,A
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (A) ← (direct) V (A)
Contoh:
Mula-mula data 9Eh (10011110b) diisikan ke RAM internal alamat 70h dengan memodifikasi isi dari RAM internal 70h, Akumulator berisi 4Fh (01001111b). Instruksi berikut:
XRL 70h,A akan menyebabkan RAM internal alamat 70h berisi data 0D1h. Untuk memonitor isi RAM Internal 70h digunakan fasilitas Wacthes yang tersedia pada DT51D. Lihat gambar24.
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0070h dan datanya adalah 70h.
Untuk memonitor isi dari RAM internal alamat 70h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0070h.

Gambar 24. Instruksi XRL direct,A pada DT51D

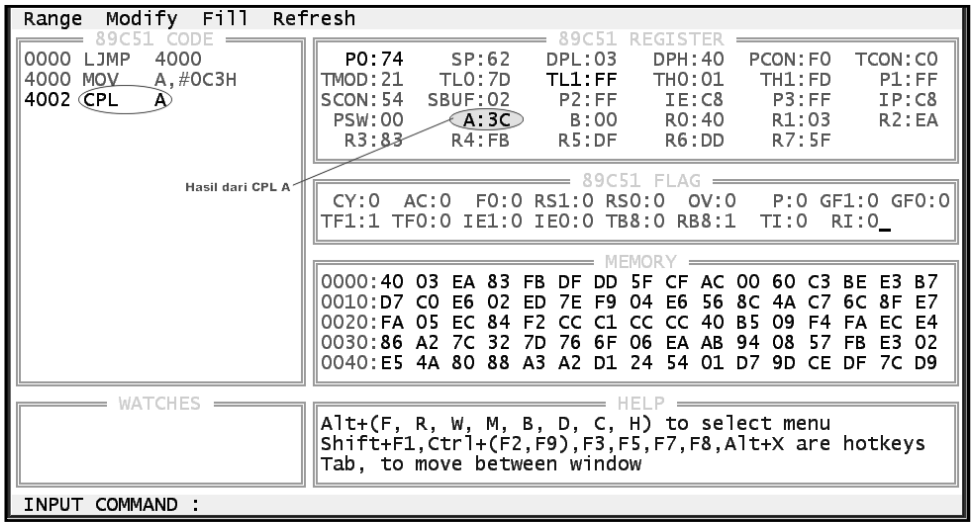


XRL direct,#data
Jumlah byte: 3 Jumlah cycle: 2
Operasi: (A) ← (direct) ⊻ #data

CPL A

Fungsi : Komplemen Accumulator
Deskripsi:
CPL A berfungsi untuk mengkomplemen tiap bit dalam Akumulator (dengan one's complement). Bit yang mula-mula bernilai 1 akan diubah menjadi 0, demikian pula sebaliknya. Tidak ada flag yang terpengaruh oleh perintah ini.
Contoh:
Mula-mula Akumulator berisi 0C3h (11000011b) diberi instruksi **CPL A** maka isi dari Accumulator saat ini adalah 03Ch (00111100b). Lihat gambar 25.
Jumlah byte: 1 Jumlah cycle: 1
Operasi: (A) ← ⌝(A)

Gambar 25. Instruksi CPL A pada DT51D



CLR A

Fungsi : Clear Akumulator
Deskripsi:
CLR A berfungsi untuk meng-clear Akumulator (Akumulator ← 0). Tidak ada flag yang dipengaruhi perintah ini.
Contoh:
Mula-mula Akumulator berisi 5Ch (01011100b). Perintah berikut ini :
CLR A
akan menghasilkan 00h (00000000b) pada Akumulator.

Jumlah byte: 1 Jumlah cycle: 1
Operasi: (A) ← 0

RL A

Fungsi: Rotasi Akumulator ke kiri
Deskripsi:
RL A berfungsi untuk merotasi kedelapan bit dalam Akumulator ke arah kiri. Bit 7 akan dipindahkan ke posisi bit 0. Tidak ada flag yang terpengaruh oleh perintah ini.
Contoh:
Akumulator mula-mula berisi 0C5h (11000101b). Perintah berikut ini :
RL A
menyebabkan Akumulator berisi 8Bh (10001011b), tanpa mempengaruhi Carry Flag.

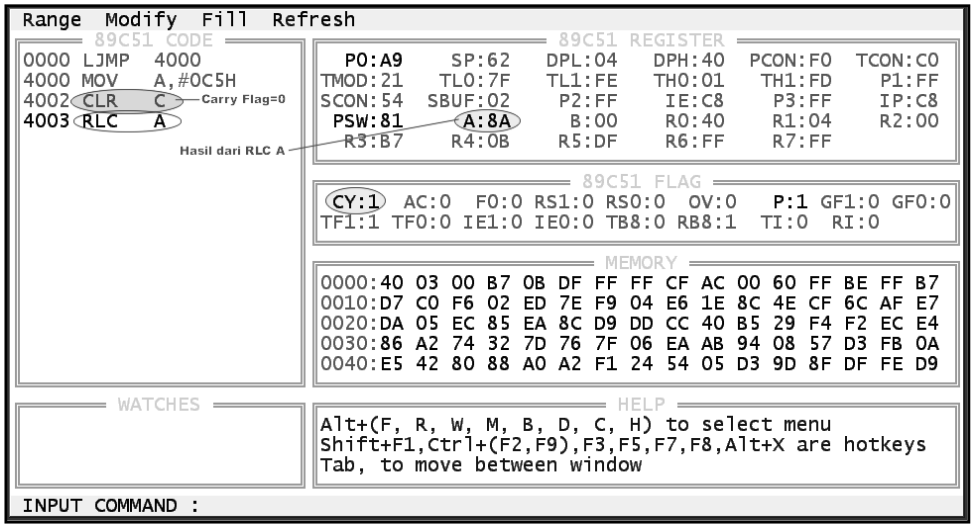
Jumlah byte: 1 Jumlah cycle: 1
Operasi: $(A_{n+1}) \leftarrow (A_n)$ n = 0-6
(A0) ← (A7)

RLC A

Fungsi : Rotasi Akumulator ke kiri melalui Carry flag
Deskripsi:
RLC A berfungsi untuk merotasi kedelapan bit dalam Akumulator ke kiri, melalui Carry Flag. Bit 7 dipindahkan ke Carry Flag, sedangkan isi Carry Flag mula-mula dipindahkan ke posisi bit 0. Tidak ada flag yang terpengaruh oleh perintah ini.
Contoh:
Akumulator mula-mula berisi 0C5h (11000101b), dan Carry Flag bernilai 0. Perintah berikut:
RLC A
menyebabkan Akumulator saat ini berisi 8Ah (10001011b) dan Carry Flag dalam keadaan set (1). Lihat gambar 26

Jumlah byte: 1 Jumlah cycle: 1
Operasi: $(A_{n+1}) \leftarrow (A_n)$ n = 0-6
(A0) ← (C)
(C) ← (A7)

Gambar 26. Instruksi RLC A pada DT51D



RR A

Fungsi : Rotasi Akumulator ke kanan

Deskripsi:

RR A berfungsi untuk merotasi kedelapan bit dalam Akumulator ke arah kanan. Bit 0 akan dipindahkan ke posisi bit 7. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Akumulator berisi 0C5h (11000101b). Perintah

RR A

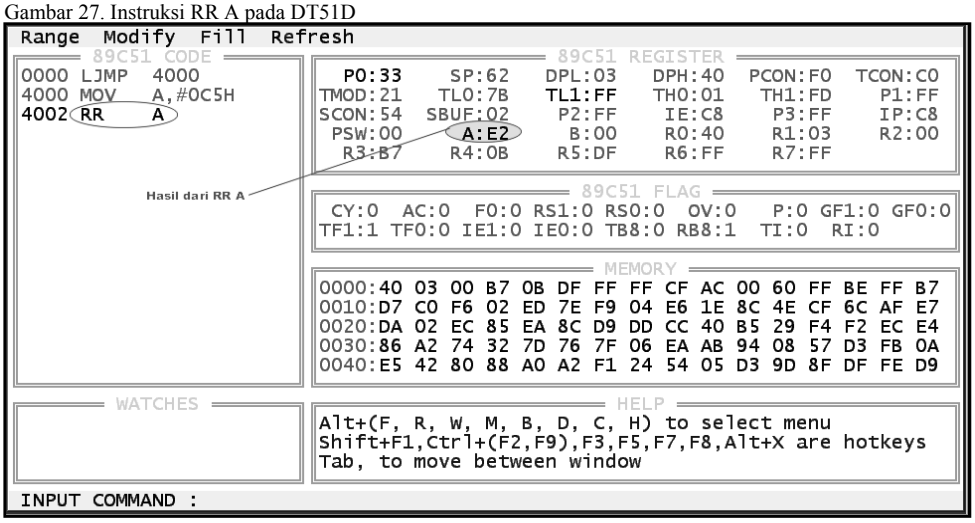
menyebabkan Akumulator berisi 0E2h (11100010b), tanpa mempengaruhi Carry Flag. Lihat gambar 27

Jumlah byte: 1 Jumlah cycle: 1

Operasi: $(A_n) \leftarrow (A_{n+1})$ n = 0-6

(A7) ← (A0)

Gambar 27. Instruksi RR A pada DT51D



RRC A

Fungsi : Rotasi Akumulator ke kanan melalui Carry Flag

Deskripsi:

RRC A berfungsi untuk merotasi kedelapan bit dalam Akumulator ke kanan, melalui Carry Flag. Bit 0 dipindahkan ke Carry Flag, sedangkan isi Carry Flag mula-mula dipindahkan ke posisi bit 7. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Akumulator bernilai 0C5h (11000101b), dan Carry Flag bernilai 0. Perintah berikut ini :

RRC A

menyebabkan Akumulator bernilai 62 (01100010h) dan Carry Flag = 1.

Jumlah byte: 1 Jumlah cycle: 1

Operasi: $(A_n) \leftarrow (A_{n+1})$ n = 0-6

(A7) ← (C)

(C) ← (A0)

SWAP A

Fungsi: Menukar posisi nibble dalam Akumulator [(A_{3-0}) ditukar dengan (A_{7-4})]

Deksripsi:

SWAP A berfungsi untuk menukar posisi low-nibble dengan high-nibble dalam Akumulator [(A3-0) ditukar dengan (A7-4)]. Operasi ini juga dapat dipandang sebagai perintah rotasi 4-bit. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Akumulator bernilai 0C5h (11000101b). Perintah berikut ini :

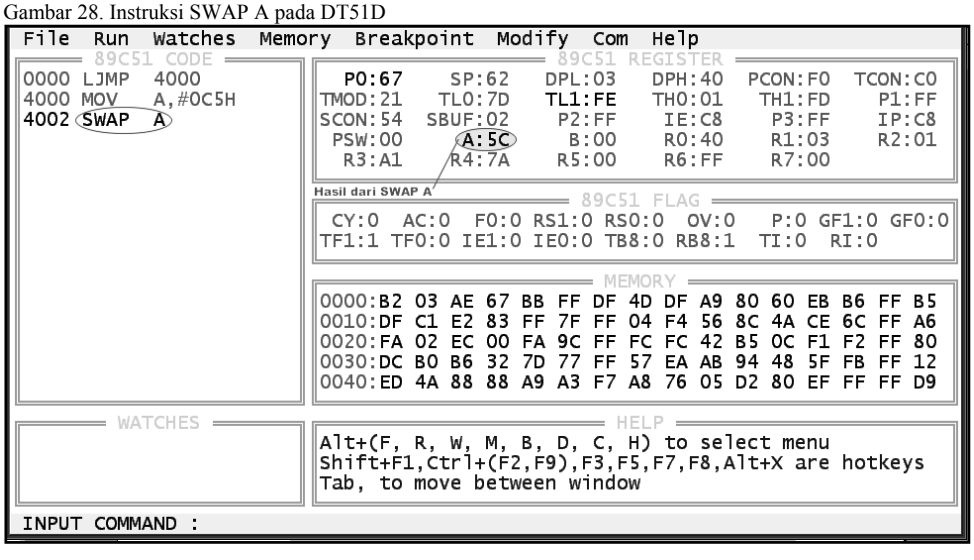
SWAP A

menyebabkan Akumulator bernilai 5Ch (01011100b). Lihat gambar 28

Jumlah byte: 1 Jumlah cycle: 1

Operasi: $(A_{3-0}) \leftrightarrow (A_{7-4})$

Gambar 28. Instruksi SWAP A pada DT51D



ANL C,<src-bit>

Fungsi: Meng-AND-kan suatu variabel bit

Deskripsi:

Perintah ini berfungsi untuk meng-AND-kan suatu variabel bit (src-bit) dengan Carry Flag, dan meletakkan hasilnya di Carry Flag. Tanda slash (/) di depan src-bit menandakan bahwa nilai yang akan di-AND-kan dengan Carry Flag adalah komplemen dari src-bit

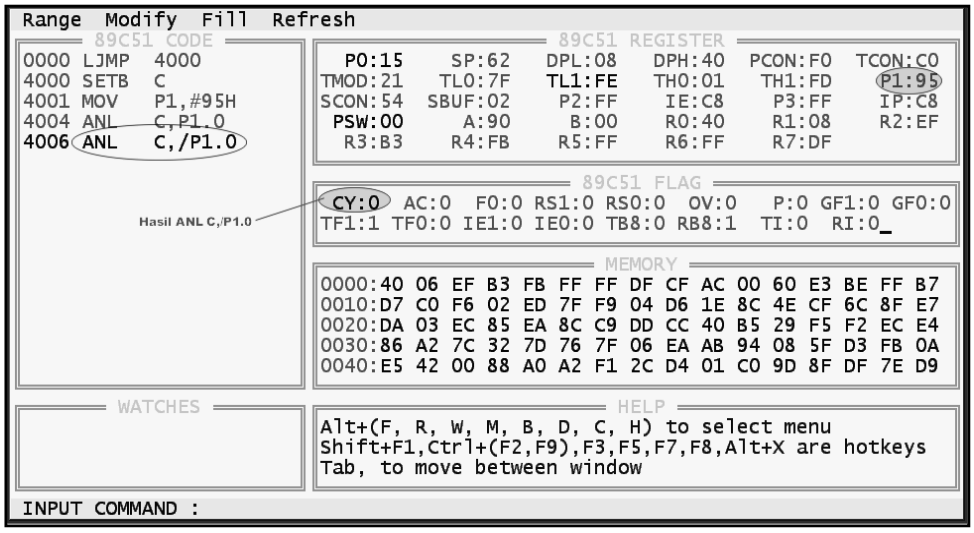
tersebut, dan bukannya src-bit itu sendiri. Nilai src-bit yang asli tetap tidak berubah. Tidak ada flag yang terpengaruh oleh perintah ini.
Hanya ada 1 mode addressing yang dapat diterapkan di sini, yaitu direct addressing.
Contoh:
"Jika P1.0 = 1, ACC.7 = 1, dan OV = 0, maka set Carry Flag menjadi 1" diimplementasikan sebagai :

```
MOV C,P1.0        ;  copy P1.0 ke Carry Flag
ANL C,ACC.7       ;  AND-kan Akumulator bit-7 dengan Carry Flag
ANL C,/OV         ; AND-kan Carry Flag dengan inverse dari Overflow Flag.
```

ANL C,bit
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (C) ← (C) Λ (bit)

ANL C,/bit
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (C) ← (C) Λ ⌝ (bit)
Contoh:
Mula-mula Carry Flag di-set menjadi 1, Port 1 berisi 95h (10010101b). Perintah berikut:
ANL C,/P1.0
menyebabkan Carry Flag berisi reset (0). Lihat gambar 29

Gambar 29. Instruksi ANL C,/bit pada DT51D



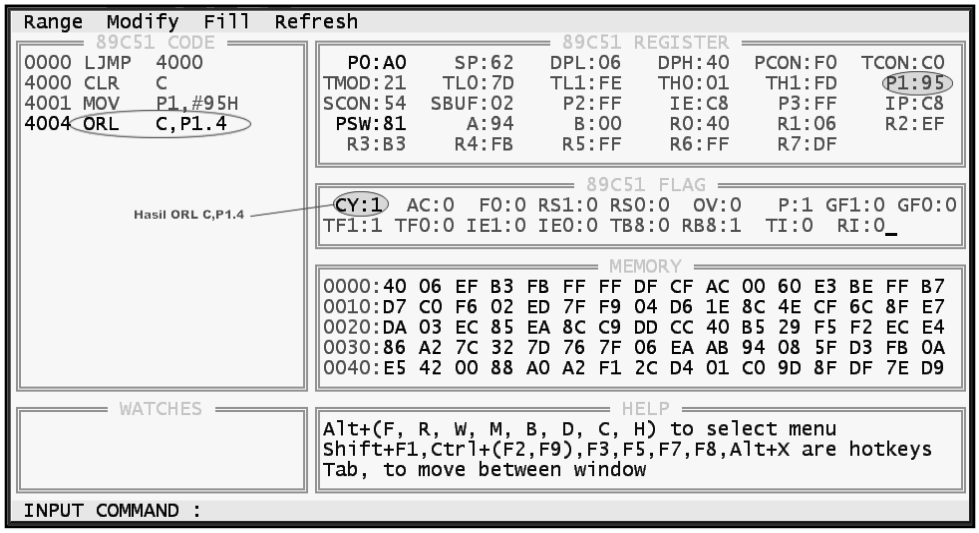
ORL C,<src-bit>

Fungsi : Meng-OR-kan variabel bit
Deskripsi:
Perintah ini berfungsi untuk meng-OR-kan suatu variabel bit (src-bit) dengan Carry Flag, dan meletakkan hasilnya di Carry Flag. Tanda slash (/) di depan src-bit menandakan bahwa nilai yang akan di-OR-kan dengan Carry Flag adalah komplemen dari src-bit tersebut, dan bukannya src-bit itu sendiri. Nilai src-bit yang asli tetap tidak berubah. Tidak ada flag yang terpengaruh oleh perintah ini.
Contoh:
"Jika P1.0 = 1, ACC. 7 = 1, or OV = 0 maka set Carry Flag menjadi 1" dapat diterapkan sebagai berikut :

```
MOV C,P1.0        ; Isi Carry Flag dengan nilai P1.0
ORL C,ACC.7       ; OR-kan Carry Flag dengan Acc.7
ORL C,/OV         ; OR-kan Carry Flag dengan inverse dari OV
```

ORL C,bit
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (C) ← (C) V (bit)
Contoh:
Mula-mula Carry Flag di-reset menjadi "0", Port 1 berisi 95h (10010101b) kemudian diberi instruksi **ORL C,P1.4** maka isi Carry Flag berubah menjadi set (1). Lihat gambar30.

Gambar 30. Instruksi ORL C,bit pada DT51D



ORL C,/bit
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (C) ← (C) V ⌝ (bit)

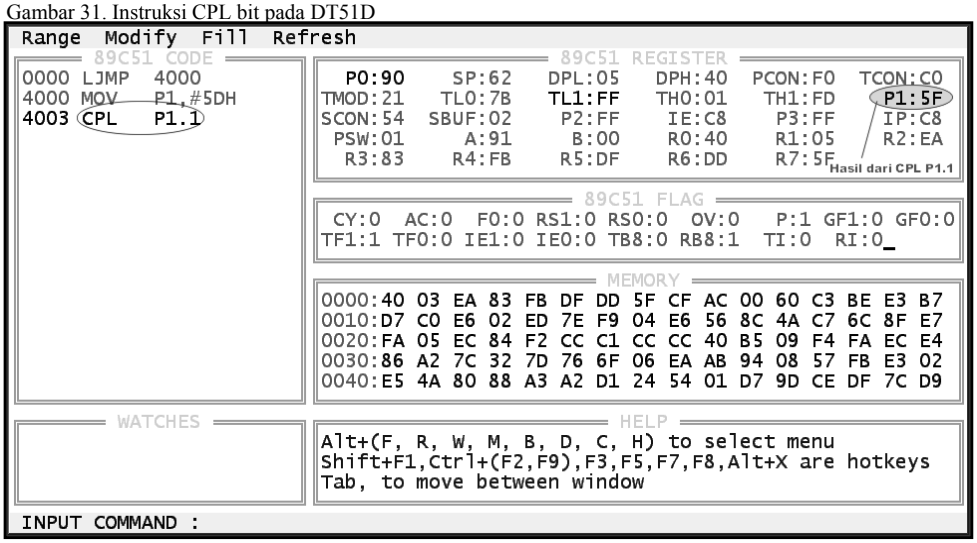
CPL <bit>

Fungsi : Komplemen bit
Deskripsi:
CPL bit berfungsi untuk mengkomplemen suatu bit. Suatu bit yang mula-mula bernilai 1 akan diubah menjadi 0, demikian pula sebaliknya. CLR hanya dapat diterapkan pada Carry Flag dan lokasi RAM internal yang bersifat bit-addressable.
Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.
Contoh:
Port 1 mula-mula bernilai 5Bh (01011101b). Perintah berikut ini :
CPL P1.1
CPL P1.2
akan menghasilkan 5Bh (01011011b) pada Port 1.

CPL C
Jumlah byte: 1 Jumlah cycle: 1
Operasi: (C) ← ⌝(C)

CPL bit
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (bit) ← ⌝(bit)
Contoh:
Port 1 mula-mula bernilai 5Dh (01011101b) kemudian diberi instruksi **CPL P1.1** maka nilai dari Port 1 saat ini adalah 5Fh (01011111b). Lihat Gambar 31

Gambar 31. Instruksi CPL bit pada DT51D



CLR <bit>

Fungsi : Clear bit

Deskripsi:

CLR bit berfungsi untuk meng-clear suatu bit tertentu (bit ← 0). Tidak ada flag lain yang terpengaruh. Perintah ini hanya dapat diterapkan pada Carry Flag dan RAM internal yang bersifat bit-addressable.

Contoh:

Port 1 mula-mula bernilai 5Dh (01011101b). Perintah berikut ini :

CLR P1.2

akan menyebabkan Port 1 bernilai 59h (01011001b).

CLR C

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (C) ← 0

CLR bit

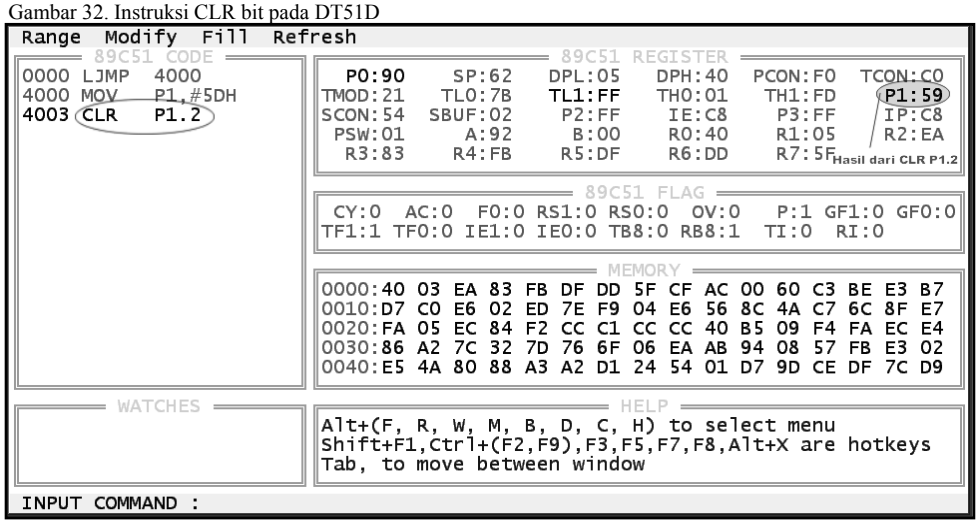
Jumlah byte: 2 Jumlah cycle: 1

Operasi: (bit) ← 0

Contoh:

Port 1 mula-mula bernilai 5Dh (01011101b) kemudian diberi instruksi **CPL P1.2** maka nilai dari Port 1 saat ini adalah 59h (01011001b). Lihat Gambar 32

Gambar 32. Instruksi CLR bit pada DT51D



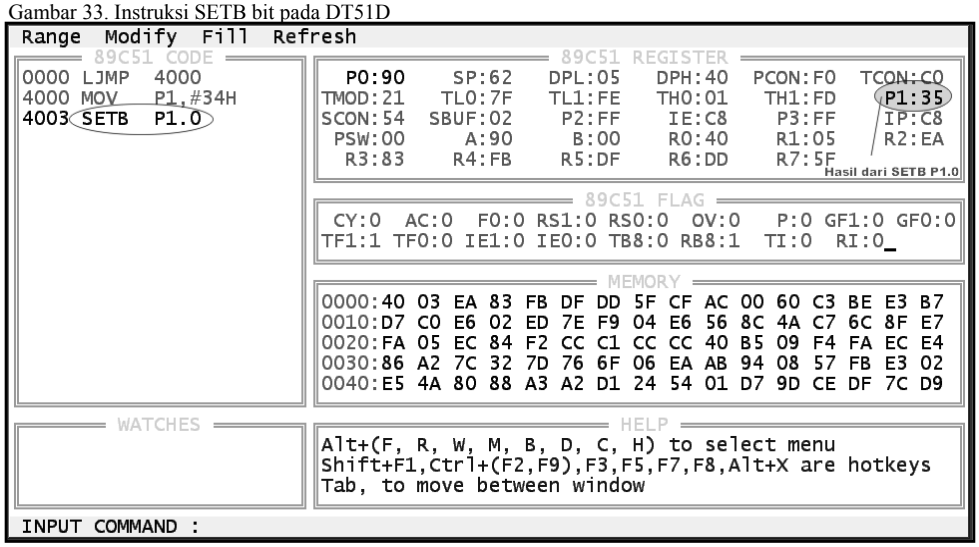
SETB <bit>

Fungsi : Set Bit
Deskripsi:
SETB bit berfungsi untuk men-set suatu bit tertentu (bit ← 1). Tidak ada flag lain yang terpengaruh. Perintah ini hanya dapat diterapkan pada Carry Flag dan RAM internal yang bersifat bit-addressable.
Contoh:
Mula-mula Carry Flag bernilai 0, dan Port 1 bernilai 34h (00110100b). Perintah berikut ini :
SETB C
SETB P1.0
menyebabkan Carry Flag bernilai 1 dan Port 1 bernilai 35h (00110101b).

SETB C
Jumlah byte: 1 Jumlah cycle: 1
Operasi: (C) ← 1

SETB bit
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (bit) ← 1
Contoh:
Port 1 mula-mula bernilai 34h (00110100b) kemudian diberi instruksi **SETB P1.0** maka isi dari Port 1 saat ini adalah 35h (00110101b). Lihat Gambar 33

Gambar 33. Instruksi SETB bit pada DT51D

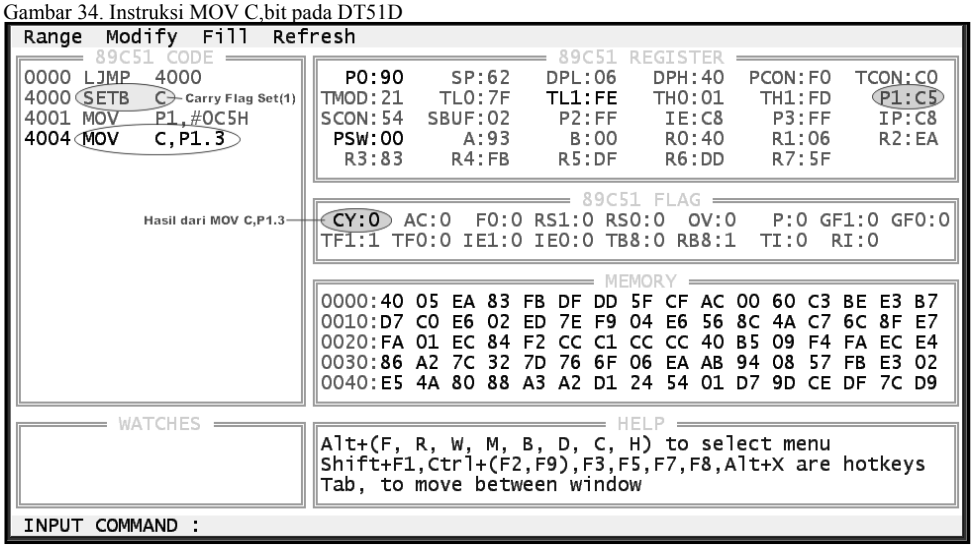


MOV <dest-bit>,<src-bit>

Fungsi: Melakukan pemindahan data antar variabel bit
Deskripsi:
MOV <dest-bit>,<src-bit> berfungsi untuk memindahkan nilai src-bit ke dest-bit, dengan tetap tidak mengubah nilai src-bit mula-mula. Salah satu operand harus berupa Carry Flag (MOV C, … atau MOV …,C). Tidak ada flag lain (selain Carry Flag) yang terpengaruh oleh perintah ini.
Contoh:
Mula-mula Carry Flag bernilai 1, Port 3 bernilai 11000101b, dan Port 1 bernilai 35h (00110101b). Perintah berikut ini :
MOV P1.3,C
MOV C,P3.3
MOV P1.2,C
menyebabkan Carry Flag di-clear menjadi 0 dan Port 1 bernilai 39h (00111001b).

MOV C,bit
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (C) ← (bit)
Contoh:
Mula-mula Carry Flag bernilai 1, Port 1 bernilai 0C5h (11000101b). Perintah berikut:
MOV C,P1.3
menyebabkan Carry Flag di-clear menjadi 0 dan Port 1 bernilai 0C5h (11000101b). Lihat Gambar 34

Gambar 34. Instruksi MOV C,bit pada DT51D



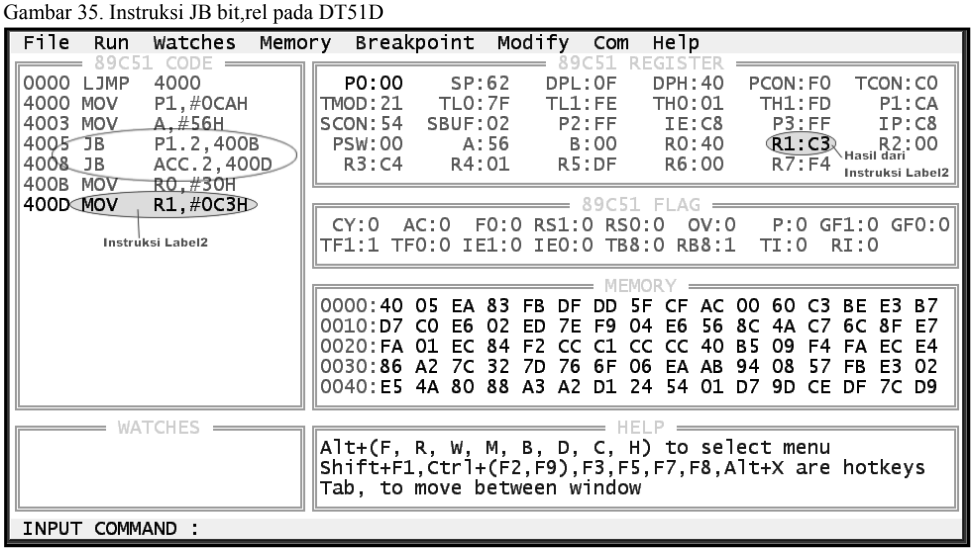
MOV bit,C
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (bit) ← (C)

JB bit,rel

Fungsi: Lompat jika bit = 1
Deskripsi:
Jika bit bernilai 1, maka program akan melompat ke alamat tujuan yang disebutkan. Jika bit bernilai 0, program akan melanjutkan ke perintah selanjutnya (tidak melakukan pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Tidak ada flag yang dipengaruhi oleh perintah ini.
Contoh:
Mula-mula port 1 bernilai 11001010b, dan Akumulator berisi 56h (01010110b). Perintah berikut ini :
JB P1.2,LABEL1
JB ACC. 2,LABEL2
menyebabkan program melompat ke alamat yang ditunjukkan oleh LABEL2. Lihat Gambar 35

Jumlah byte: 3 Jumlah cycle: 2
Operasi: (PC) ← (PC) + 3
IF (bit) = 1
THEN
(PC) ← (PC) + rel

Gambar 35. Instruksi JB bit,rel pada DT51D



JBC bit,rel

Fungsi: Lompat jika bit = 1 dan clear bit tersebut

Deskripsi:

Jika bit bernilai 1, maka JBC akan melompat ke alamat yang disebutkan; jika tidak maka program akan melanjutkan ke perintah selanjutnya (tidak terjad pelompatan). Selain itu, nilai bit tersebut akan di-clear menjadi 0. Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Tidak ada flag yang dipengaruhi oleh perintah ini.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

Mula-mula Akumulator bernilai 56h (01010110b). Perintah berikut ini :

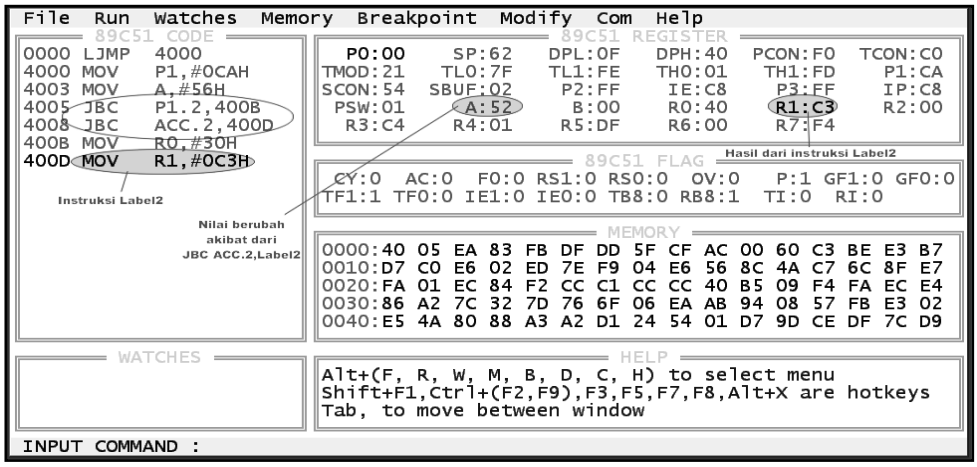
JBC ACC.3,LABEL1
JBC ACC.2,LABEL2

menyebabkan program melompat ke alamat yang ditunjukkan oleh LABEL2, dan nilai Akumulator diubah menjadi 52h (01010010b). Lihat Gambar 36

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
IF (bit) = 1
THEN
(bit) ← 0
(PC) ← (PC) +rel

Gambar 36. Instruksi JBC bit,rel pada DT51D



JC rel

Fungsi: Lompat jika Carry Flag = 1

Deskripsi:

Jika Carry Flag = 1, maka program akan melompat ke alamat yang disebutkan dalam perintah; jika tidak, maka program akan melanjutkan ke baris berikutnya (tidak terjadi pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Tidak ada flag yang dipengaruhi oleh perintah ini.

Contoh:

Mula-mula Carry Flag = 0. Perintah berikut ini :

JC LABEL1

CPL C

JC LABEL 2

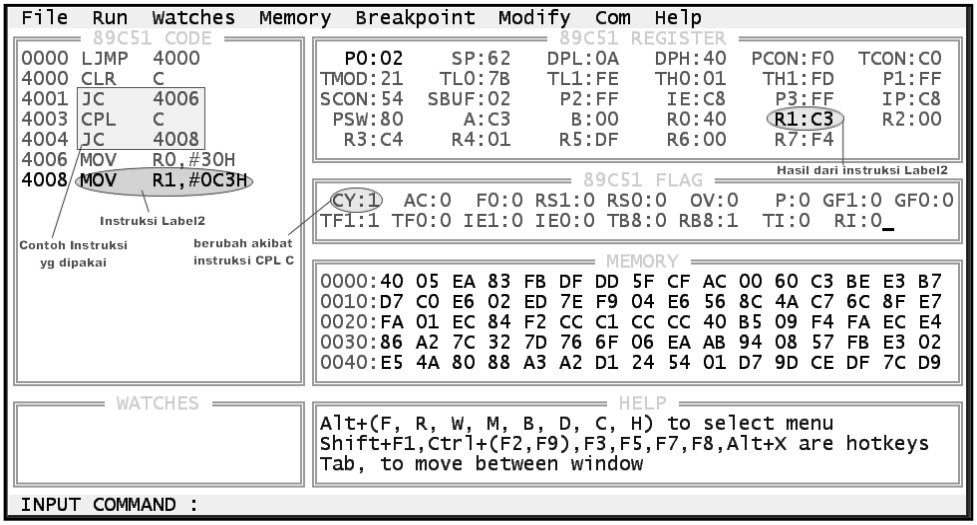
akan men-set Carry Flag menjadi 1 dan menyebabkan program melompat ke alamat yang ditunjukkan oleh LABEL2. Lihat Gambar 37

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2

IF (C) = 1 THEN (PC) ← (PC) + rel

Gambar 37. Instruksi JC rel pada DT51D



JNB bit,rel

Fungsi: Lompat jika bit = 0

Deskripsi:

Jika bit bernilai 0, maka program akan melompat ke alamat tujuan yang disebutkan. Jika bit bernilai 1, program akan melanjutkan ke perintah selanjutnya (tidak melakukan pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Tidak ada flag yang dipengaruhi oleh perintah ini.

Contoh:

Mula-mula port 1 bernilai 11001010b, dan Akumulator bernilai 56h (01010110b). Perintah berikut ini :

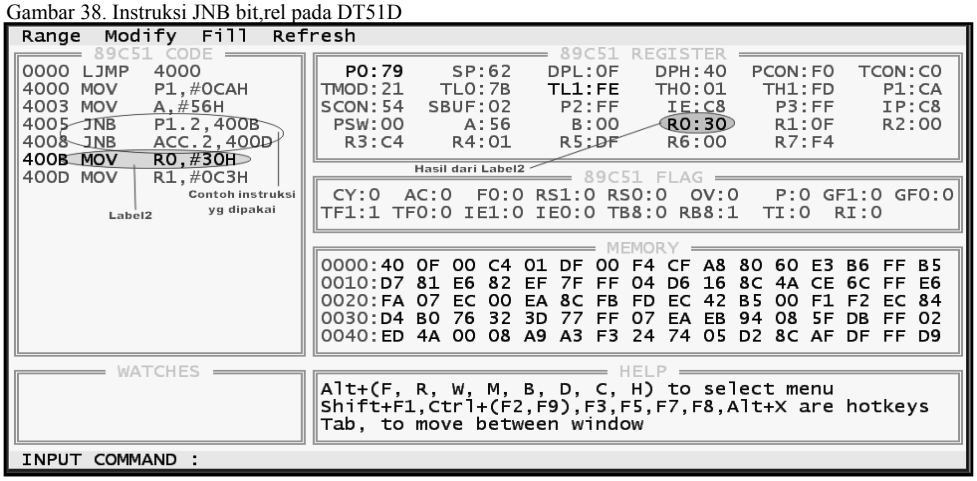
JNB P1.3,LABEL1
JNB ACC.3,LABEL2

menyebabkan program melompat ke alamat yang ditunjukkan oleh LABEL2. Lihat Gambar 38.

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
IF (bit) = 0 THEN (PC) ← (PC) + rel

Gambar 38. Instruksi JNB bit,rel pada DT51D



JNC rel

Fungsi: Lompat jika Carry Flag = 0

Deskripsi:

Jika Carry Flag = 0, maka program akan melompat ke alamat yang disebutkan dalam perintah; jika tidak, maka program akan melanjutkan ke baris berikutnya (tidak terjadi pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Tidak ada flag yang dipengaruhi oleh perintah ini.

Contoh:

Mula-mula Carry Flag bernilai 1. Perintah berikut ini :

JNC LABEL1
CPL C
JNC LABEL2

akan meng-clear Carry Flag menjadi 0 dan menyebabkan program melompat ke alamat yang ditunjukkan oleh LABEL2.

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
IF (C) = 0
THEN
(PC) ← (PC) + rel

C. TRANSFER DATA

Macam-macam instruksi Transfer Data dapat dilihat di tabel 4

Tabel 4. Instruksi Transfer Data

Mnemonic & Operand	Operasi	Execution Time (μs)
MOV A,<src>	A = <src>	1
MOV <dest>,A	<dest> = A	1
MOV <dest>,<src>	<dest> = <src>	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant	2
PUSH <src>	INC SP : MOV "@SP",<src>	2
POP <dest>	MOV <dest>,"@SP" : DEC SP	2
XCH A,<byte>	ACC dan <byte> saling menukar data	1
XCHD A,@Ri	ACC dan @Ri saling menukar low-nibbles	1

MOV <dest-byte>,<src-byte>

Fungsi: Melakukan pemindahan data antar variabel byte

Deskripsi:

Ketika perintah MOV dieksekusi, nilai src-byte akan dipindahkan ke dest-byte. Nilai src-byte sendiri tidak berubah setelah eksekusi. Tidak ada flag yang terpengaruh oleh perintah ini.

Terdapat 15 kombinasi addressing yang dapat diterapkan pada dest-byte dan src-byte.

Contoh:

Mula-mula RAM internal alamat 30h berisi data 40h, sedangkan alamat 40h berisi data 10h. Port 1 bernilai 11001010b (0CAh).

```
MOV R0,#30H
MOV A,@R0
MOV R1,A
MOV B,@R1
MOV @R1,P1
MOV P2,P1
```

Perintah di atas menyebabkan R0 = 30h, Akumulator = 40h, R1 = 40h, B = 10h, RAM internal alamat 40h = 0CAh (11001010b), port 2 = 0CAh.

MOV A,Rn

Jumlah byte: 1 Jumlah cycle:1

Operasi: (A) ← (Rn)

MOV A,direct

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← (direct)

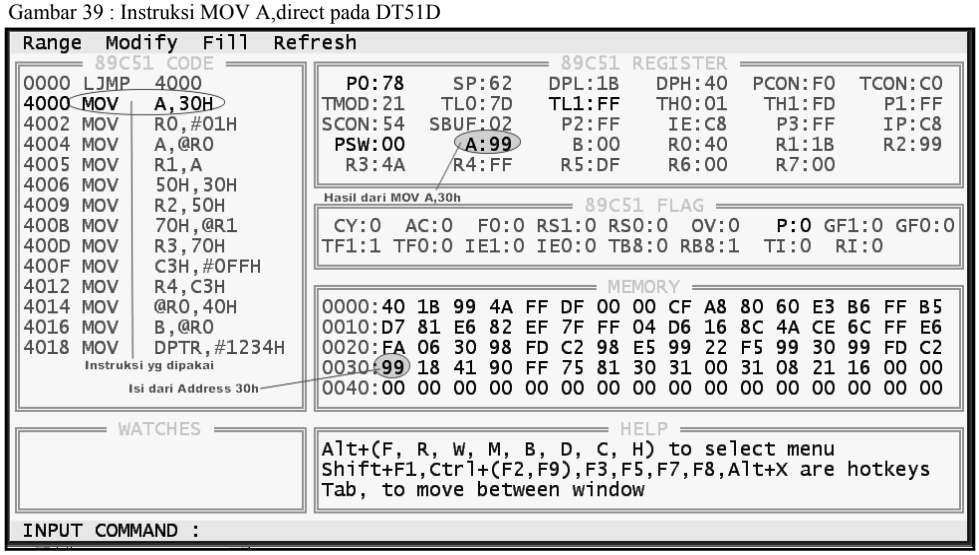
Instruksi MOV A,Acc tidak diperbolehkan!

Contoh:

Mula-mula data 99h diisikan ke RAM internal alamat 30h dengan memodifikasi isi dari memori alamat 30h. Perintah **MOV A,30h** menyebabkan Akumulator berisi data 99h. Lihat Gambar 39

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0030h dan datanya adalah 99h.

Gambar 39 : Instruksi MOV A,direct pada DT51D



MOV A,@Ri

Jumlah byte: 1 Jumlah cycle: 1

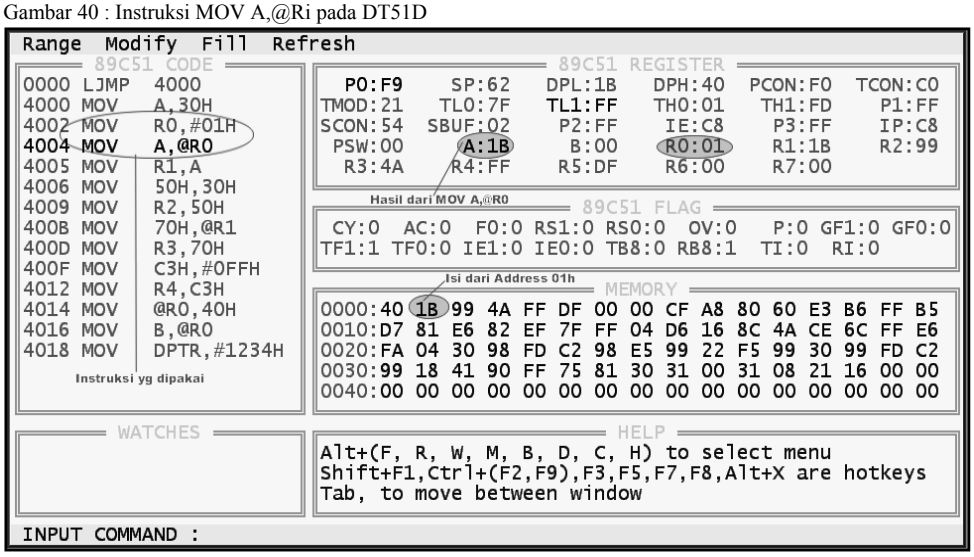
Operasi: (A) ← ((Ri))

Contoh:

Mula-mula Register R0 berisi 01h (00000001b), data 18h diisikan ke RAM internal alamat 01h dengan memodifikasi isi memori 01h. Perintah **MOV A,@R0** menyebabkan Akumulator berisi data 18h. Lihat Gambar 40

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0001h dan datanya adalah 18h.

Gambar 40 : Instruksi MOV A,@Ri pada DT51D



MOV A,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: (A) ← #data

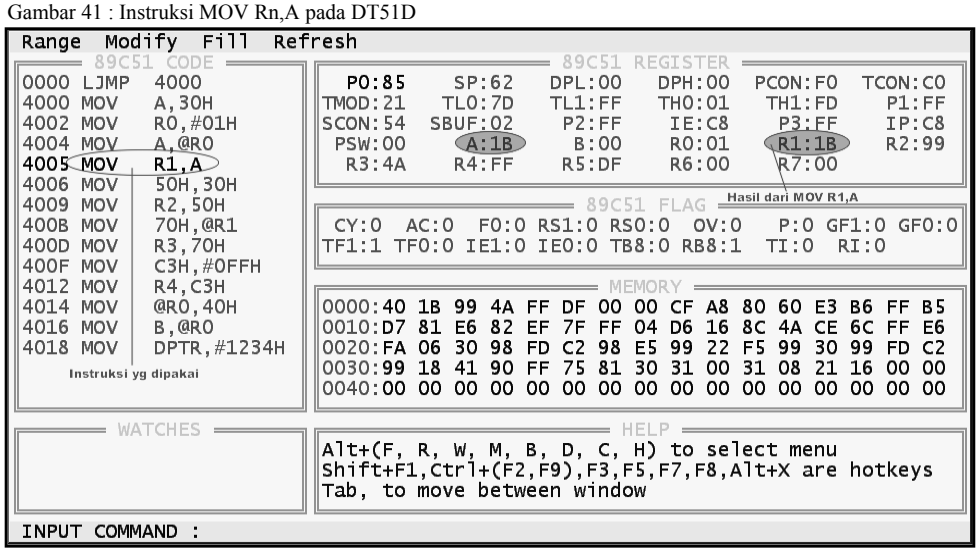
MOV Rn,A

Jumlah byte: 1 Jumlah cycle:1

Operasi: (Rn) ← (A)

Contoh:
Mula-mula Akumulator berisi 18h (00011000b). Perintah **MOV R1,A** menyebabkan Register R1 berisi 18h (00011000b). (lihat Gambar 41)

Gambar 41 : Instruksi MOV Rn,A pada DT51D



MOV Rn,direct
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (Rn) ← (direct)

MOV Rn,#data
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (Rn) ← #data

MOV direct,A
Jumlah byte: 2 Jumlah cycle: 1
Operasi: (direct) ← (A)

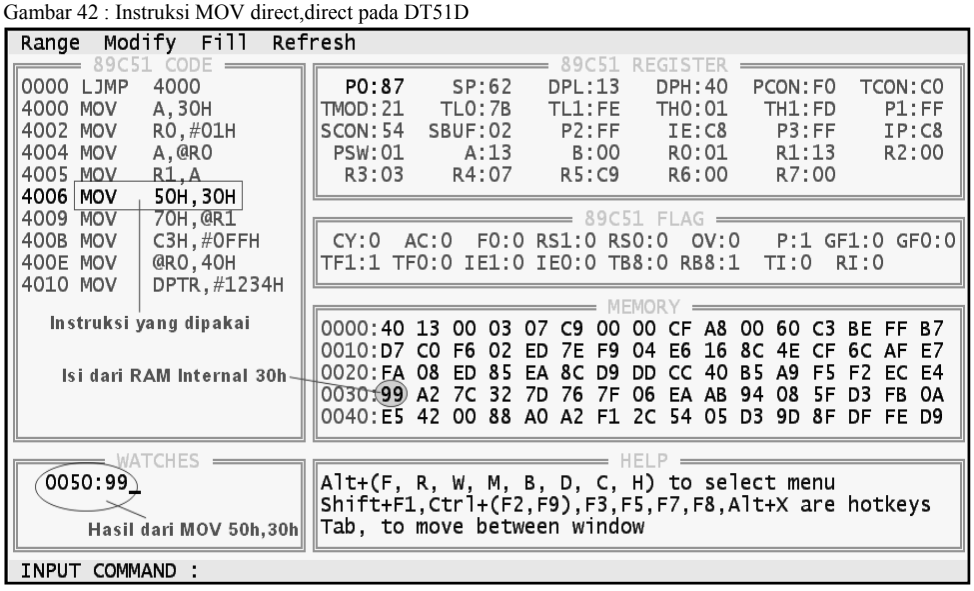
MOV direct,Rn
Jumlah byte: 2 Jumlah cycle: 2
Operasi: (direct) ← (Rn)

MOV direct,direct
Jumlah byte: 3 Jumlah cycle: 2
Operasi: (direct) ← (direct)
Contoh:
Mula-mula data 99h diisikan ke RAM internal alamat 30h dengan memodifikasi memori 30h. Perintah **MOV 50h,30h** menyebabkan RAM internal alamat 50h berisi data 99h. Untuk memonitor hasil dari instruksi ini dapat digunakan fasilitas Watches yang terdapat pada DT51D. Lihat Gambar42.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0030h dan datanya adalah 99h.

Untuk memonitor isi dari RAM internal alamat 50h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0050h.

Gambar 42 : Instruksi MOV direct,direct pada DT51D



MOV direct,@Ri

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (direct) ← ((Ri))

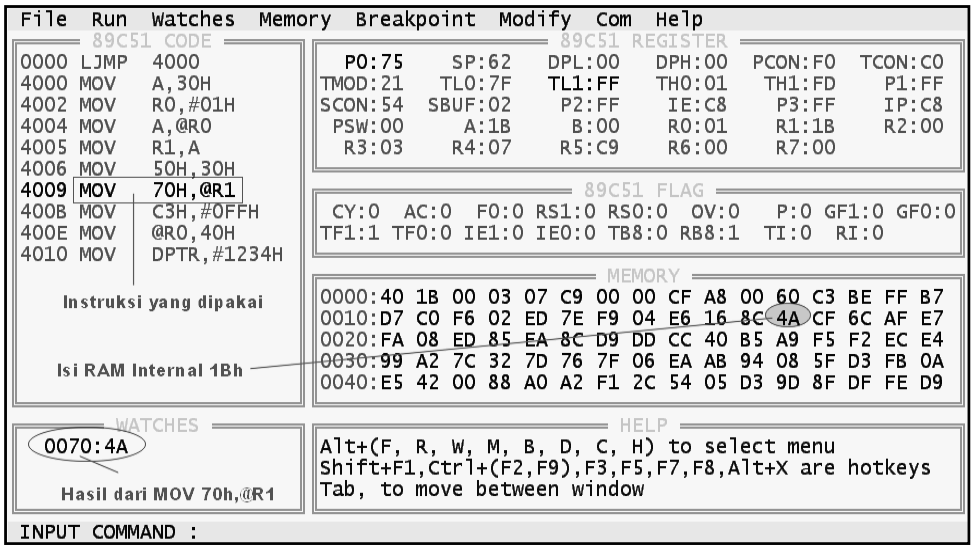
Contoh:

Mula-mula Register R1 berisi 1Bh (00011011b), data 4Ah diisikan ke RAM internal alamat 1Bh dengan memodifikasi memori 1Bh. Perintah **MOV 70h,@R1** menyebabkan RAM internal alamat 70h berisi data 4Ah. Hasil dari instruksi ini dapat dimonitor dengan fasilitas Wacthes yang tersedia pada DT51D.Lihat Gambar 43

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 001Bh dan datanya adalah 4Ah.

Untuk memonitor isi dari RAM internal alamat 70h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0070h.

Gambar 43 : Instruksi MOV direct,@Ri pada DT51D



MOV direct,#data

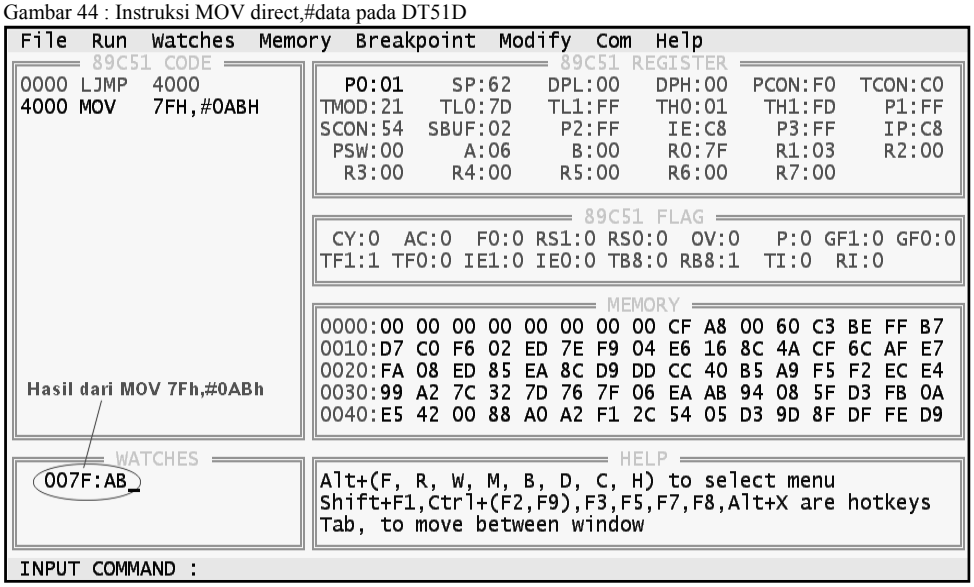
Jumlah byte: 3 Jumlah cycle: 2

Operasi: (direct) ← #data

Contoh:
Perintah **MOV 7Fh,#0ABh** menyebabkan RAM internal alamat 7Fh berisi data 0ABh. Untuk memonitor isi dari RAM Internal alamat 7Fh digunakan fasilitas Watches yang tersedia pada DT51D. Lihat Gambar 44

Catatan: Untuk memonitor isi dari RAM internal alamat 7Fh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 007Fh.

Gambar 44 : Instruksi MOV direct,#data pada DT51D

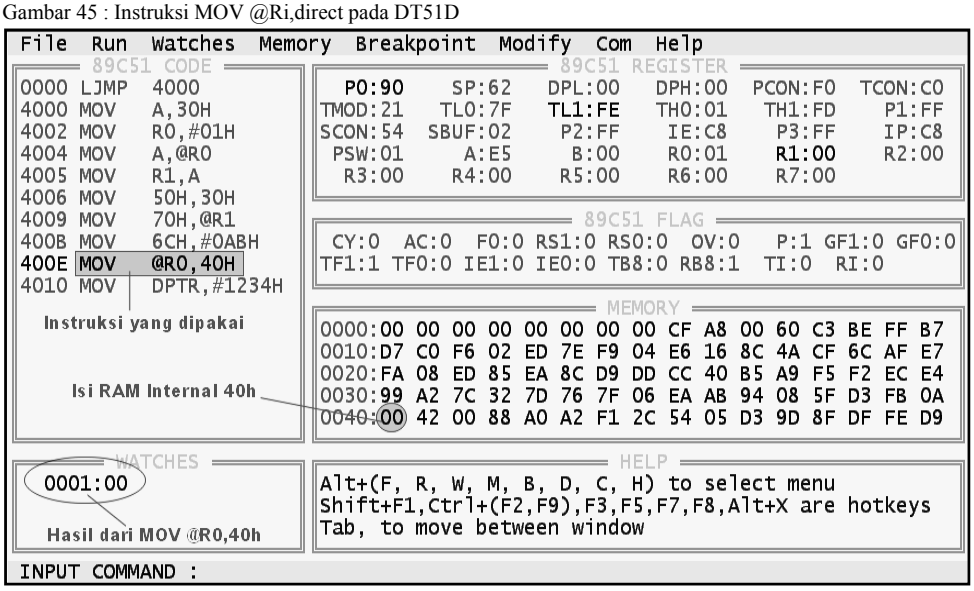


MOV @Ri,A
Jumlah byte: 1 Jumlah cycle: 1
Operasi: ((Ri)) ← (A)

MOV @Ri,direct
Jumlah byte: 2 Jumlah cycle: 2
Operasi: ((Ri)) ← (direct)
Contoh:
Mula-mula Register R1 berisi 01h (00000001b) kemudian data 00h (00000000b) diisikan ke RAM internal alamat 40h dengan memodifikasi isi memori 40h. Perintah **MOV @R0, 40h** menyebabkan RAM internal alamat 40h berisi data 00h. Hasil dari instruksi ini (alamat 01h) dapat dimonitor dengan fasilitas Watches yang tersedia pada DT51D. Lihat Gambar 45

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0040h dan datanya adalah 00h.
Untuk memonitor isi dari RAM internal alamat 01h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adala alamat 0001h.

Gambar 45 : Instruksi MOV @Ri,direct pada DT51D



MOV @Ri,#data

Jumlah byte: 2 Jumlah cycle: 1

Operasi: ((Ri)) ← #data

MOV DPTR,#data16

Fungsi: Mengisi Data Pointer (DPTR) dengan konstanta 16-bit.

Deskripsi:

MOV DPTR,#data16 berfungsi untuk mengisi DPTR dengan suatu konstanta 16-bit. Konstanta ini diperoleh dari byte ke-2 dan ke-3 yang mengikuti perintah DPTR. Byte ke-2 merupakan high-byte dan diisikan ke DPH, sedangkan byte ke-3 merupakan low-byte dan diisikan ke DPL. Tidak ada flag yang terpengaruh oleh perintah ini.

Perintah ini merupakan satu-satunya perintah yang dapat melakukan pemindahan data 16-bit.

Contoh:

Perintah **MOV DPTR,#1234h** menyebabkan DPTR bernilai 1234h, dengan komposisi DPH=12h , dan DPL=34h. Lihat Gambar 46

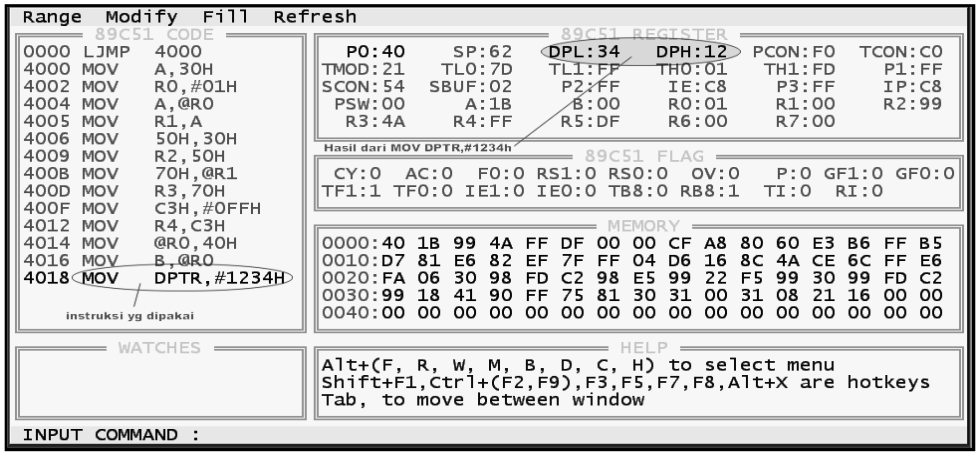
Jumlah byte: 3 Jumlah cycle: 2

Operasi: (DPTR) ← $\#data_{15\text{-}0}$

DPH ← $\#data_{15\text{-}8}$

DPL ← $\#data_{7\text{-}0}$

Gambar 46 : Instruksi MOV DPTR,#data16 pada DT51D



MOVC A,@A+ <base-reg>

Fungsi: Mengambil suatu kode dari memori program.

Deskripsi:

MOVC berfungsi untuk mengambil suatu kode dari memori program dan meletakkannya dalam Akumulator. Alamat yang menunjukkan lokasi kode yang akan diambil diperoleh dengan menjumlahkan nilai Akumulator dengan base-reg. Base-reg dapat berupa DPTR atau PC.

Jika base-reg berupa PC, maka sebelum penjumlahan di atas dilakukan nilai PC terlebih dahulu di-increment untuk mendapatkan alamat perintah berikutnya yang mengikuti MOVC. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Akumulator berisi bilangan dari 0 s.d. 3. Perintah berikut ini akan mengisi Akumulator dengan salah satu dari keempat bilangan yang didefinisikan dengan DB (define byte).

```
REL_PC:   INC A
          MOVC A,@A+PC
          RET
          DB 66h
          DB 77h
          DB 88h
          DB 99h
```

Sebagai contoh, jika Akumulator mula-mula bernilai 01h, maka pada akhir instruksi Akumulator akan bernilai 77h. Perintah INC A di atas diperlukan untuk 'melompati' perintah RET, sehingga nilai awal 0 pada Akumulator akan menyebabkan diperolehnya bilangan pertama yaitu 66h.

MOVC A,@A+DPTR

Jumlah byte: 1 Jumlah cycle: 2

Operasi: (A) ← ((A) + (DPTR))

Contoh:

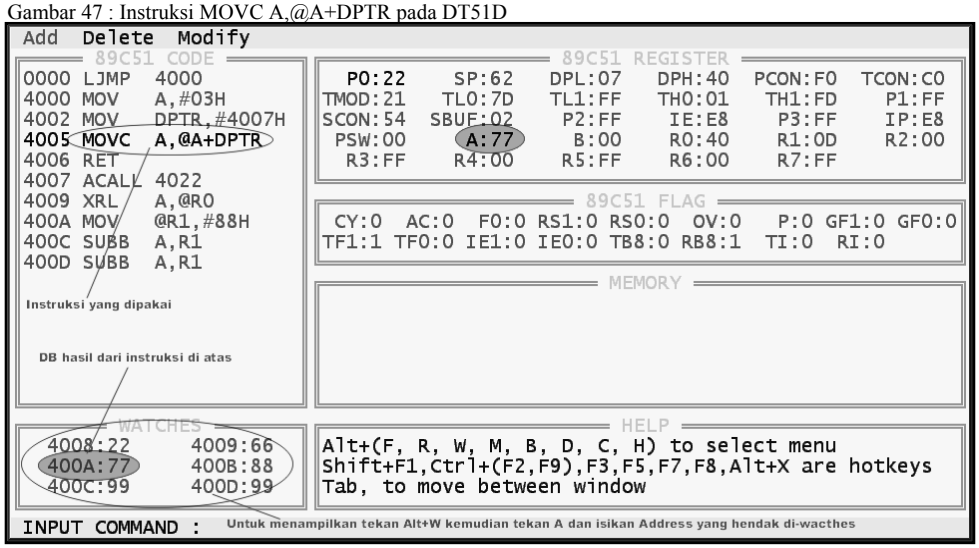
Mula-mula Akumulator berisi bilangan 0 s.d. 5 dan DPTR berisi 4007h. Perintah berikut ini akan mengisi Akumulator dengan salah satu dari keenam bilangan yang didefinisikan dengan DB (define byte).

```
MOVC A,@A+DPTR
RET
DB 11h
DB 22h
DB 66h
DB 77h
DB 88h
DB 99h
```

Sebagai contoh, jika Akumulator mula-mula bernilai 03h, maka pada akhir instruksi Akumulator akan bernilai 77h. Hal ini disebabkan isi dari Akumulator ditambah dengan nilai DPTR yang ada sehingga bilangan yang akan diisikan ke Akumulator dengan instruksi **MOVC A,@A+PC** adalah bilangan yang didefinisikan dengan alamat DB 400Ah yaitu 66h. Lihat Gambar 47

Catatan: Untuk memonitor isi dari memory alamat 4008h, 4009h, 400Ah, 400Bh, 400Ch, 400Dh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 4008h, 4009h, 400Ah, 400Bh, 400Ch, 400Dh.

Gambar 47 : Instruksi MOVC A,@A+DPTR pada DT51D



MOVC A,@A+PC

Jumlah byte: 1 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 1

(A) ← ((A) + (PC))

Contoh:

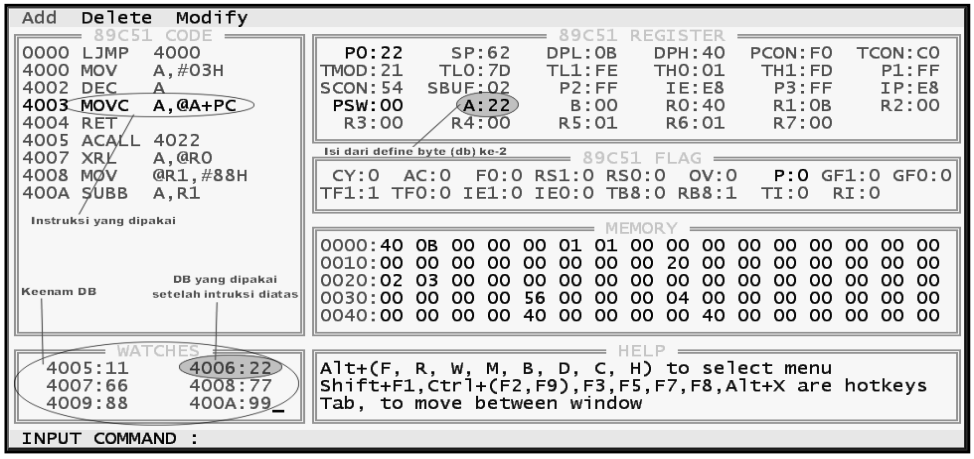
Mula-mula Akumulator berisi bilangan 0 s.d. 5. Perintah berikut ini akan mengisi Akumulator dengan salah satu dari keenam bilangan yang didefinisikan dengan DB (define byte).

```
DEC A
MOVC A,@A+PC
RET
DB 11h
DB 22h
DB 66h
DB 77h
DB 88h
DB 99h
```

Sebagai contoh, jika Akumulator mula-mula bernilai 03h, maka pada akhir instruksi Akumulator akan bernilai 22h. Hal ini disebabkan akumulator di-decrement satu kali sehingga pada saat ini Akumulator bernilai 02h, maka bilangan yang diisikan ke Akumulator dengan instruksi **MOVC A,@A+PC** adalah bilangan yang didefinisikan dengan DB urutan kedua dari atas yaitu 22h. (lihat Gambar 48)

Catatan: Untuk memonitor isi dari memory alamat 4005h, 4006h, 4007h, 4008h, 4009h, 400Ah digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 4005h, 4006h, 4007h, 4008h, 4009h, 400Ah.

Gambar 48 : Instruksi MOVC A,@A+PC pada DT51D



MOVX <dest-byte>,<src-byte>

Fungsi: Membaca/menulis data dari/ke memori eksternal..

Deskripsi:

Perintah MOVX berfungsi untuk melakukan transfer data antara Akumulator dengan suatu byte di memori eksternal. "X" dalam istilah "MOVX" berarti "eksternal". Jika Akumulator berada pada posisi dest-byte, maka operasi yang dilakukan adalah operasi pembacaan memori eksternal. Jika Akumulator berada pada src-byte, maka operasi yang dilakukan adalah operasi penulisan memori eksternal. Terdapat dua mode yang dapat digunakan di sini, yaitu MOVX dengan alamat 8-bit dan dengan alamat 16-bit.

Pada mode pertama, digunakan register-indirect addressing dengan menggunakan R0 dan R1. Alamat yang dapat digunakan di sini hanya berukuran 8-bit, dan cocok digunakan untuk ekspansi I/O atau untuk penggunaan RAM berukuran kecil. Ketika perintah ini dieksekusi, baik data maupun address 8-bit akan dimultipleks di Port 0.

Pada mode kedua, digunakan register-indirect addressing dengan menggunakan DPTR. Alamat yang dapat digunakan di sini berukuran 16-bit, di mana high-byte (DPH) dikeluarkan ke Port 2, sedangkan low-byte (DPL) dan data 8-bit dimultipleks di Port 0.

Contoh:

Sebuah RAM eksternal berukuran 256 byte dihubungkan ke mikrokontroler MCS-51. Port 3 berfungsi sebagai control-bus untuk external RAM ini. Mula-mula R0 berisi 12h, dan R1 berisi 34h. RAM eksternal alamat 34h berisi data 56h. Perintah berikut ini :

MOVX A,@R1
MOVX @R0,A

akan menyebabkan Akumulator dan RAM eksternal alamat 12h masing-masing berisi data 56h.

MOVX A,@Ri

Jumlah byte: 1 Jumlah cycle: 2

Operasi: (A) ← ((Ri))

MOVX A,@DPTR

Jumlah byte: 1 Jumlah cycle: 2

Operasi: (A) ← ((DPTR))

Contoh:

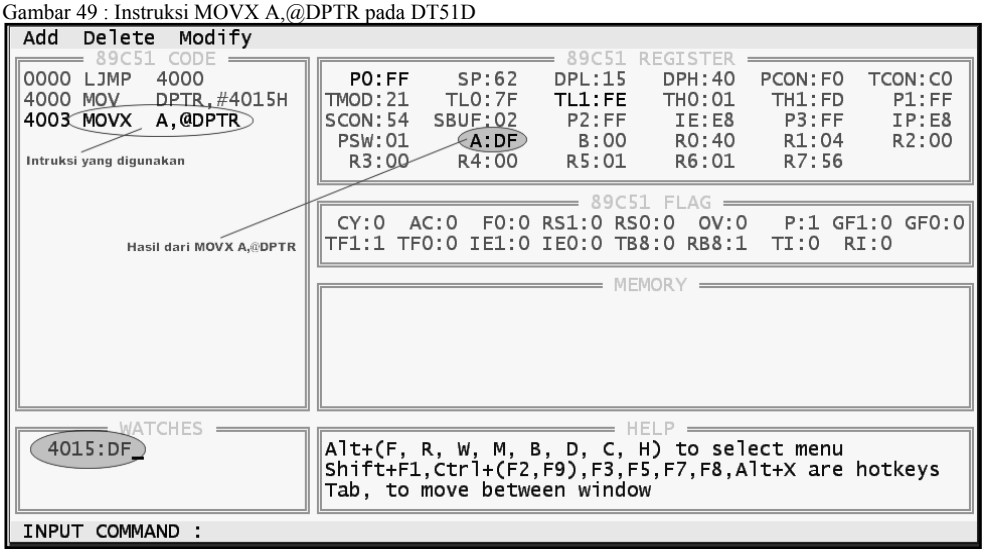
Mula-mula DPTR berisi 4015h. RAM eksternal alamat 4015h berisi DFh. Perintah berikut ini:

MOVX A,@DPTR

menyebabkan isi dari Akumulator menjadi isi dari RAM eksternal 4015h yaitu DFh. Untuk melihat isi dari RAM eksternal 4015h gunakan fasilitas Wacthes yang tersedia pada DT51D. Lihat Gambar 49

Catatan: Untuk memonitor isi dari RAM eksternal alamat 4015h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 4015h.

Gambar 49 : Instruksi MOVX A,@DPTR pada DT51D



MOVX @Ri,A

Jumlah byte: 1 Jumlah cycle: 2

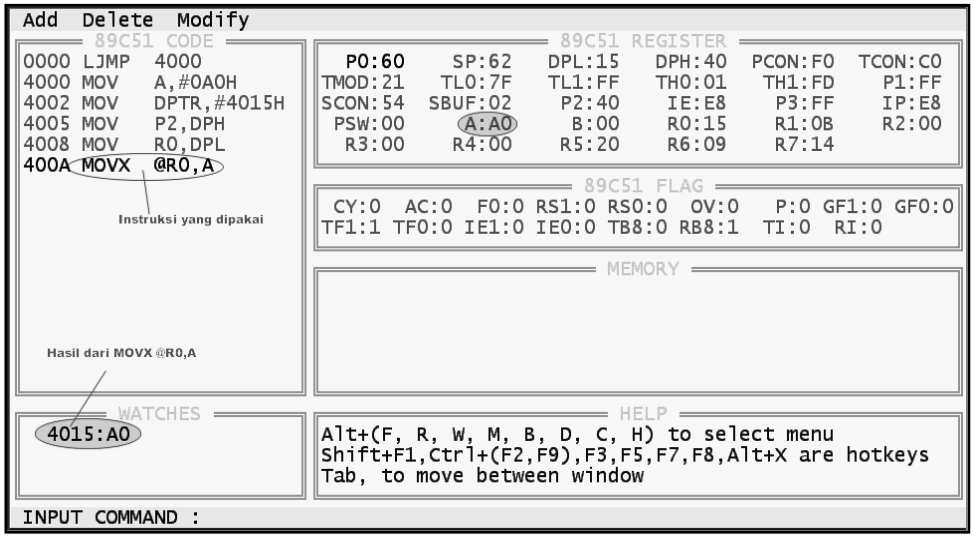
Operasi: ((Ri)) ← (A)

Contoh:

Akumulator mula-mula berisi 0A0h (10100000b), DPTR berisi 4015h. Perintah: **MOVX @R0,A** menyebabkan RAM eksternal alamat 4015h berisi 0A0h. Catatan: Untuk melihat isi dari RAM eksternal alamat 4015h gunakan fasilitas Watches yang tersedia pada DT51D. Lihat Gambar 50

Catatan: Untuk memonitor isi dari RAM ekternal alamat 4015h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 4015h.

Gambar 50: Instruksi MOVX @Ri,A pada DT51D



MOVX @DPTR,A

Jumlah byte: 1 Jumlah cycle: 2

Operasi: ((DPTR)) ← (A)

POP direct

Fungsi: Pop dari stack.

Deskripsi:

Isi RAM internal yang ditunjukkan oleh Stack Pointer dibaca, dan nilai Stack Pointer di-decrement. Hasil pembacaan ini selanjutnya ditransfer ke variabel byte yang disebutkan dalam perintah. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Stack Pointer mula-mula bernilai 09h, sedangkan RAM internal alamat 8h-9h berturut-turut bernilai 20h, dan B0h. Perintah berikut ini :

POP 01h

POP 02h

menyebabkan Stack Pointer bernilai 07 dan isi dari RAM internal alamat 01h-02h berturut-turut bernilai B0h dan 20h. Untuk memonitor isi dari RAM internal 01h, dan 02h digunakan fasilitas Watches yang dimiliki DT51D. Lihat Gambar 51

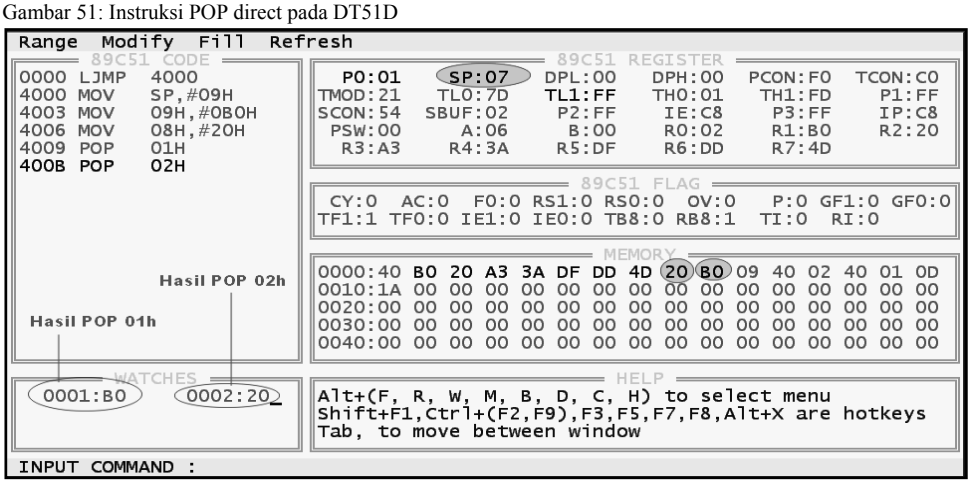
Catatan: Untuk memonitor isi dari RAM internal alamat 01h, 02h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0001h, 0002h.
Untuk mengamati hasil dari alamat 09h, dan 08h dapat me-refresh memory setelah perintah POP dieksekusi.

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (direct) ← ((SP))

(SP) ← (SP) – 1

Gambar 51: Instruksi POP direct pada DT51D



PUSH direct

Fungsi: Push ke stack.

Deskripsi:

Stack Pointer di-increment, lalu isi variabel byte yang disebutkan dalam perintah ditransfer ke RAM internal pada lokasi yang ditunjukkan oleh Stack Pointer. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Stack Pointer bernilai 09h, dan DPTR bernilai 0123h. Perintah berikut ini :

PUSH DPL

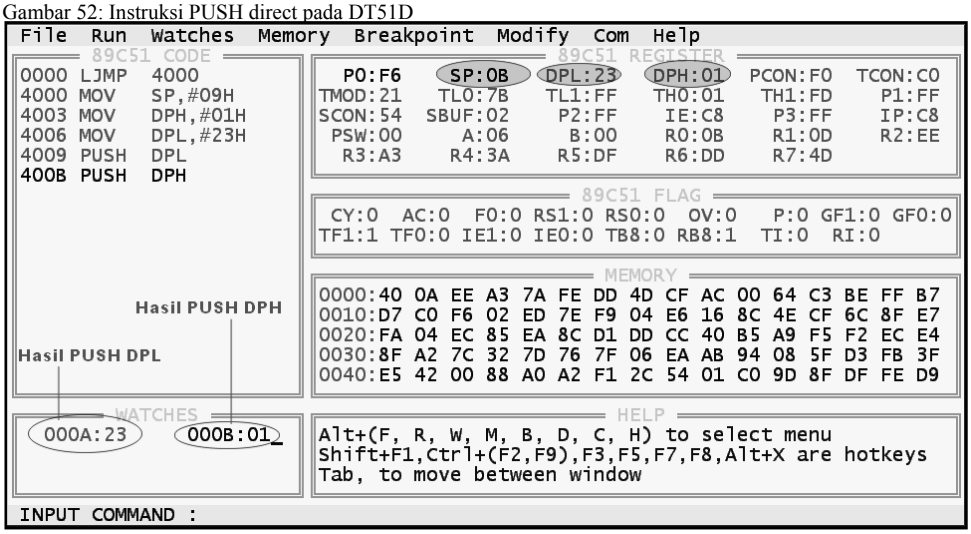
PUSH DPH

menyebabkan Stack Pointer bernilai 0Bh dan RAM internal alamat 0Ah dan 0BH berturut-turut bernilai 23h dan 01h. Untuk memonitor isi dari RAM Internal alamat 0Ah dan 0Bh gunakan fasilitas Watches yang tersedia pada DT51D.Lihat Gambar 52

Catatan: Untuk memonitor isi dari RAM internal alamat 0Ah dan 0Bh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 000Ah dan 000Bh.

Jumlah byte: 2 Jumlah cycle: 2
Operasi: (SP) ← (SP) + 1
((SP)) ← (direct)

Gambar 52: Instruksi PUSH direct pada DT51D



XCH A,<byte>

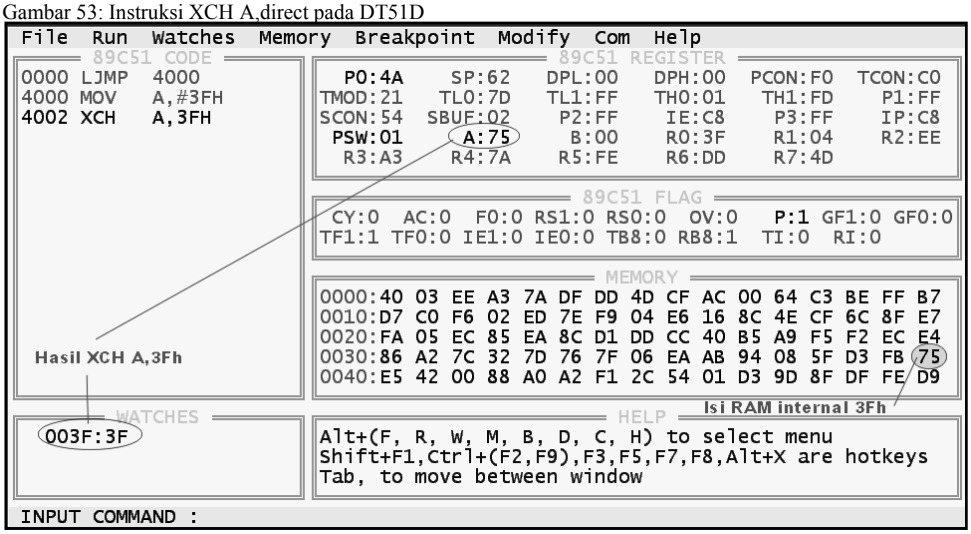
Fungsi: Menukar isi Akumulator dengan isi variabel byte.
Deskripsi:
XCH berfungsi untuk menukar isi Akumulator dengan isi suatu variabel byte. Variabel byte yang digunakan dapat berupa register, direct, atau register-indirect addressing.
Contoh:
Mula-mula R0 berisi 20h, Akumulator berisi 3Fh (0011111lb), RAM internal alamat 20h berisi 75h (01110101b). Perintah berikut ini :
XCH A,@R0
menyebabkan RAM internal alamat 20h berisi data 3Fh (00111111b), dan Akumulator berisi 75h (01110101b).

XCH A,Rn
Jumlah byte: 1 Jumlah cycle:1
Operasi: (A) ⇔ (Rn)

XCH A,direct
Jumlah byte: 2 Jumlah cycle:1
Operasi: (A) ⇔ (direct)

Contoh:
Akumulator mula-mula berisi 3Fh (00111111b), data 75h (01110101b) diisikan ke RAM internal alamat 3Fh dengan memodifikasi isi RAM internal alamat 3Fh. Perintah **XCH A,3Fh** menyebabkan Akumulator berisi 75h (01110101b), dan RAM internal alamat 3Fh berisi data 3Fh (00111111b). Isi dari RAM internal alamat 3Fh dapat dimonitor dengan menggunakan fasilitas Watches yang terdapat pada DT51D. Lihat Gambar 53
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 003Fh dan datanya adalah 75h.
Untuk memonitor isi dari RAM internal alamat 3Fh digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 003Fh.

Gambar 53: Instruksi XCH A,direct pada DT51D



XCH A,@Ri

Jumlah byte: 1 Jumlah cycle:1

Operasi: (A) ⇔ ((Ri))

XCHD A,@Ri

Fungsi: Exchange Digit

Deskripsi:

XCHD berfungsi untuk menukar low-nibble (bit 3-0) Akumulator dengan low-nibble suatu variabel byte yang terletak dalam RAM internal. High-nibble (bit 7-4) Akumulator maupun variabel byte tersebut tidak berubah. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Register R1 berisi 30h, Akumulator berisi 0FFh (11111111b), dan data 88h (10001000b) diisikan ke RAM internal alamat 30h dengan memodifikasi isi dari RAM internal 30h. Perintah berikut ini :

XCHD A,@R0

menyebabkan RAM internal alamat 30h berisi data 8Fh (10001111b) dan Akumulator berisi 0F8h (11111000b). Untuk memonitor isi dari alamat 30h digunakan fasilitas Watches yang tersedia pada DT51D. Lihat Gambar 54

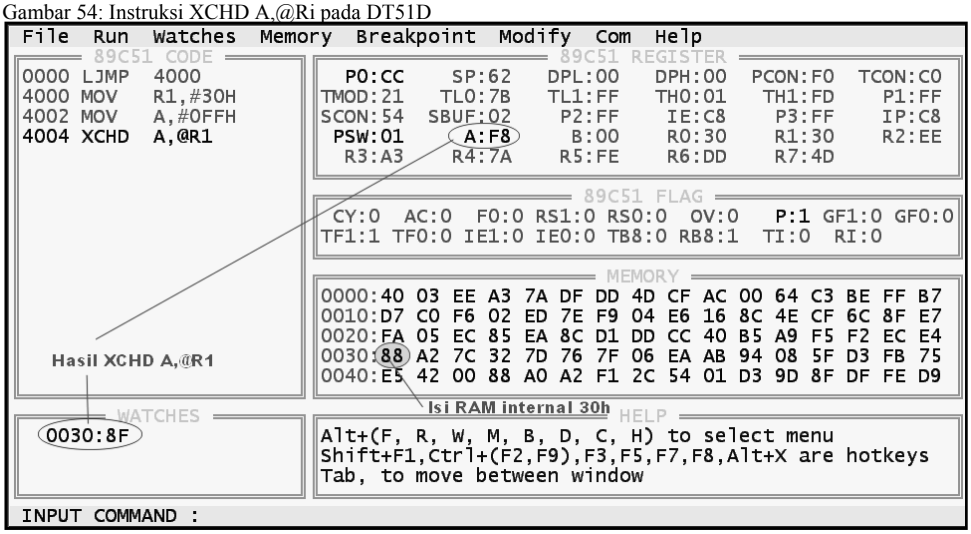
Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0030h dan datanya adalah 88h.

Untuk memonitor isi dari RAM internal alamat 30h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0030h.

Jumlah byte: 1 Jumlah cycle: 1

Operasi: (A_{3-0}) ⇔ ((Ri_{3-0}))

Gambar 54: Instruksi XCHD A,@Ri pada DT51D



D. PERCABANGAN DALAM PROGRAM

ACALL addr11

Fungsi: Memanggil sub-rutin (absolute call).

Deskripsi:

ACALL berfungsi untuk memanggil sub-rutin yang terdapat pada alamat tertentu. Instruksi ini akan meng-increment PC 2 kali untuk mendapatkan alamat perintah berikutnya yang mengikuti ACALL. Setelah itu alamat 16-bit tersebut di-push ke stack (byte rendah terlebih dahulu), dan nilai Stack Pointer (SP) di-increment 2 kali.

Alamat sub-rutin tujuan diperoleh dengan cara menggabungkan bit 15-11 dari PC (yang sudah di-increment seperti dijelaskan di atas), opcode bit 7-5, dan byte kedua (operand) yang mengikuti perintah ACALL. Karena bit 15-11 selalu didapatkan dari PC (yang sudah di-increment), maka alamat tujuan harus berada dalam blok 2 KByte yang sama (bit 15-11–nya sama) dengan perintah berikutnya yang mengikuti ACALL. Tidak ada flag yang dipengaruhi oleh perintah ini.

Contoh:

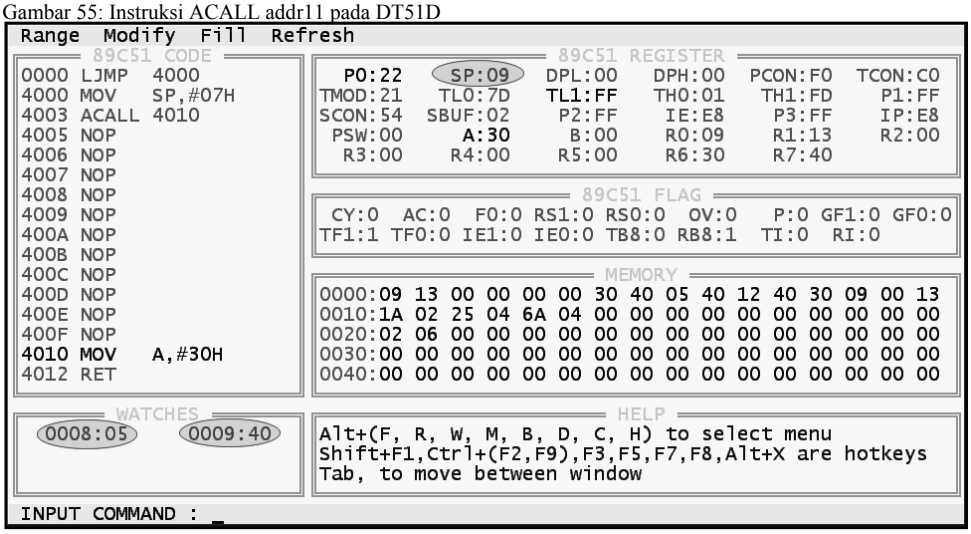
Mula-mula SP bernilai 07H, dan subrutin "CONTROL" berada pada alamat 4010h. Perintah di bawah ini berada pada alamat 4003h. Setelah perintah tersebut dijalankan :

ACALL CONTROL

maka SP akan bernilai 09h, RAM internal alamat 08h-09h akan berturut-turut bernilai 05h dan 40h, sedangkan PC akan bernilai 4003h. Lihat Gambar 55

Catatan: Untuk memonitor isi dari RAM internal alamat 08h, dan 09h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0008h dan 0009h.

Gambar 55: Instruksi ACALL addr11 pada DT51D



Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC_{10-0}) ← alamat tujuan

LCALL addr16

Fungsi: Memanggil sub-rutin (long call).

Deskripsi:

LCALL berfungsi untuk memanggil sub-rutin pada alamat tertentu. Ketika instruksi ini dieksekusi, maka PC akan di-increment sampai didapatkan alamat perintah berikutnya

yang mengikuti LCALL. Alamat 16-bit ini di-push ke stack (low byte terlebih dahulu), dan Stack Pointer di-increment 2 kali. Setelah itu PC diisi dengan alamat tujuan (16-bit), yang diperoleh dari byte ke-2 dan ke-3 yang mengikuti perintah LCALL (byte ke-2 sebagai high-byte, byte ke-3 sebagai low-byte). Jangkauan alamat tujuan yang dapat digunakan adalah 64 KByte (0000H-FFFFH). Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Stack Pointer bernilai 07h, label CONTROL berada pada alamat 1234h, dan perintah berikut ini :

LCALL CONTROL berada pada alamat 0123h. Setelah perintah tersebut dieksekusi, maka Stack Pointer akan bernilai 09h, dan RAM internal alamat 08h dan 09h akan berturut-turut bernilai 26h dan 01h, sedangkan PC akan bernilai 1234h.

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

AJMP addr11

Fungsi: Lompat ke alamat tertentu (absolute jump).

Deskripsi:

AJMP berfungsi untuk membuat program melompat ke alamat tertentu. Instruksi ini akan meng-increment PC 2 kali untuk mendapatkan alamat perintah berikutnya yang mengikuti AJMP. Setelah itu program akan melompat ke alamat tujuan yang diperoleh dengan cara menggabungkan bit 15-11 dari PC (yang sudah di-increment), opcode bit 7-5, dan byte kedua (operand) yang mengikuti perintah AJMP. Karena bit 15-11 selalu didapatkan dari PC (yang sudah di-increment), maka alamat tujuan harus berada dalam blok 2 KByte yang sama (bit 15-11–nya sama) dengan perintah berikutnya yang mengikuti AJMP.

Contoh:

Label MIKRO terdapat pada alamat 0123h. Perintah berikut ini :

AJMP MIKRO berada pada alamat 0345h, dan jika dieksekusi akan menyebabkan PC (Program Counter) berisi 0123h.

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
(PC_{10-0}) ← alamat tujuan

LJMP addr16

Fungsi: Lompat (long jump).

Deskripsi:

LJMP menyebabkan program melompat ke alamat tertentu. Alamat tujuan ini diperoleh dari byte ke-2 dan ke-3 yang mengikuti perintah LJMP (byte ke-2 sebagai high-byte, byte ke-3 sebagai low-byte). Dengan demikian maka alamat tujuan bisa berada dalam jangkauan 64 KByte (0000h-FFFFh). Tidak ada flag yang terpengaruh oleh perintah ini.

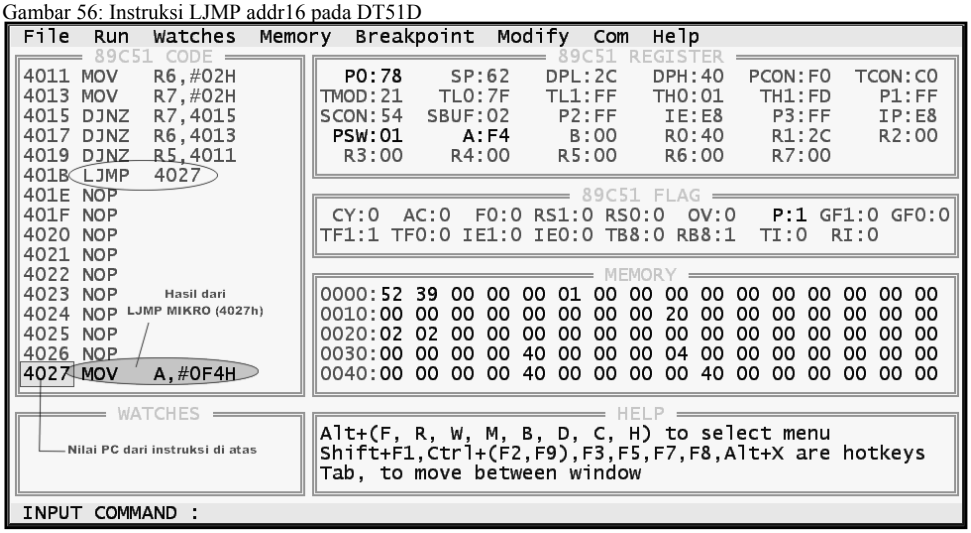
Contoh:

Label MIKRO berada pada alamat 401Bh. Perintah berikut ini :

LJMP MIKRO

berada pada alamat 4027h, dan akan menyebabkan PC bernilai 4027h. Lihat Gambar 56

Gambar 56: Instruksi LJMP addr16 pada DT51D



Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← $addr_{15-0}$

SJMP rel

Fungsi: Short Jump.

Deskripsi:

SJMP menyebabkan program melompat ke alamat tertentu. Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment 2 kali untuk memperoleh alamat perintah berikutnya yang mengikuti SJMP. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini.

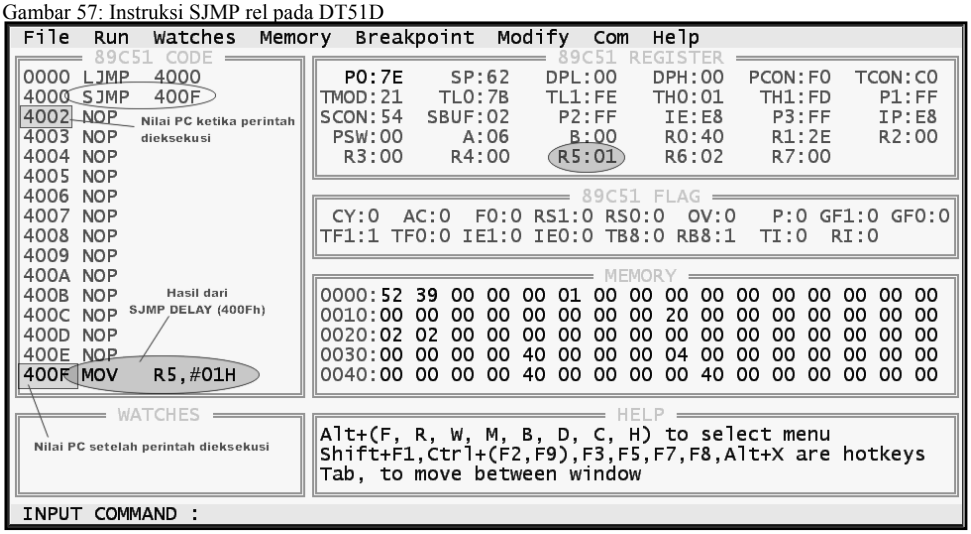
Contoh:

Label DELAY berada pada lokasi 400Fh. Perintah berikut ini :

SJMP DELAY

berada pada lokasi 4000h. Jika perintah tersebut dieksekusi, maka PC akan berisi 400Fh. Lihat Gambar 57

Gambar 57: Instruksi SJMP rel pada DT51D



Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
(PC) ← (PC) + rel

CJNE <dest-byte>,<src-byte>, rel

Fungsi: Lompat jika dest-byte <> src-byte.

Deskripsi:

CJNE akan membandingkan nilai dest-byte dengan src-byte, dan melompat ke alamat tertentu jika kedua nilai tersebut tidak sama. Ketika perintah ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti CJNE. Alamat tujuan diperoleh dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini.

Jika nilai dest-byte kurang dari src-byte (keduanya dianggap sebagai unsigned-integer), maka Carry Flag di-set menjadi 1. Jika dest-byte lebih dari atau sama dengan src-byte, maka Carry Flag di-clear menjadi 0.

Terdapat 4 mode addressing yang dapat diterapkan pada dest-byte dan src-byte. Akumulator dapat dibandingkan dengan direct byte dan immediate data. Register (Rn) dan Indirect RAM (@Ri) dapat dibandingkan immediate constant.

Contoh:

Mula-mula R7 berisi 56h. Perintah berikut ini :

```
        CJNE R7, # 60H, MIKRO
BEDA:  JC MIKRO
```

akan men-set Carry Flag menjadi 1, dan melompat ke MIKRO. Selanjutnya dengan melihat nilai Carry Flag, instruksi pada lokasi MIKRO dapat menentukan apakah R7 lebih atau kurang dari 60h.

CJNE A,direct,rel

Jumlah byte: 3 Jumlah cycle: 2

Operasi:
```
(PC) ← (PC) + 3
IF   (A) < > (direct)
THEN
     (PC) ← (PC) + relative offset
IF   (A) < (direct)
THEN
     (C) ← 1
ELSE
     (C) ← 0
```

Contoh:

Mula-mula Akumulator berisi 30h, data 40h diisikan ke RAM internal alamat 50h dengan memodifikasi isi RAM internal 50h. Label QUICK berada pada lokasi 4005h. Instruksi berikut :

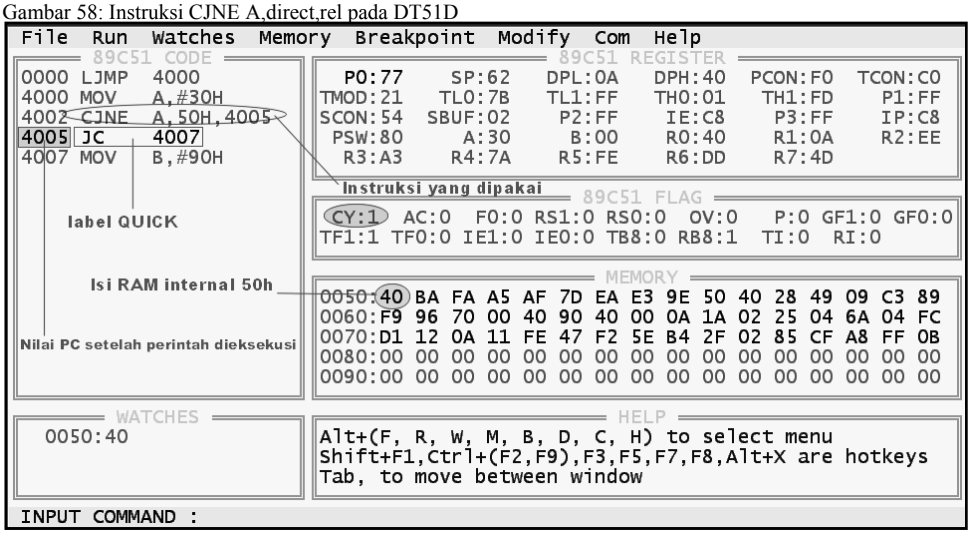
```
          CJNE A,50h,QUICK
QUICK:    JC NOW
NOW:      MOV B,#90h
```

akan men-set Carry Flag menjadi 1, dan melompat ke QUICK. Selanjutnya dengan melihat nilai Carry Flag, instruksi pada lokasi QUICK dapat menentukan apakah A lebih atau kurang dari data RAM internal alamat 50h. Apabila nilai A tidak sama dengan nilai dari RAM internal alamat 50h, maka akan melompat ke QUICK dan meneruskan ke intruksi selanjutnya. Setelah eksekusi PC akan berisi 4005h. Lihat Gambar 58.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0050h dan datanya adalah 40h. Untuk dapat menampilkan alamat 50h pada window memory, arahkan kursor ke window memory (tekan Tab sebanyak 4 kali) pada baris 0050h.

Untuk memonitor isi dari RAM internal alamat 50h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0050h.

Gambar 58: Instruksi CJNE A,direct,rel pada DT51D



CJNE A,#data,rel

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
IF (A) < > data
THEN
(PC) ← (PC) + relative offset
IF (A) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE Rn,#data,rel

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
IF (Rn) < > data
THEN
(PC) ← (PC) + relative offset
IF (Rn) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE @Ri,#data,rel

Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 3
IF ((Ri)) < > data
THEN
(PC) ← (PC) + relative offset
IF ((Ri)) < data
THEN
(C) ← 1
ELSE
(C) ← 0

Contoh:

Mula-mula register R0 berisi 40h, data 40h diisikan ke RAM internal alamat 40h dengan memodifikasi RAM internal 40h. Label JUMP berada pada lokasi 4007h. Perintah berikut:

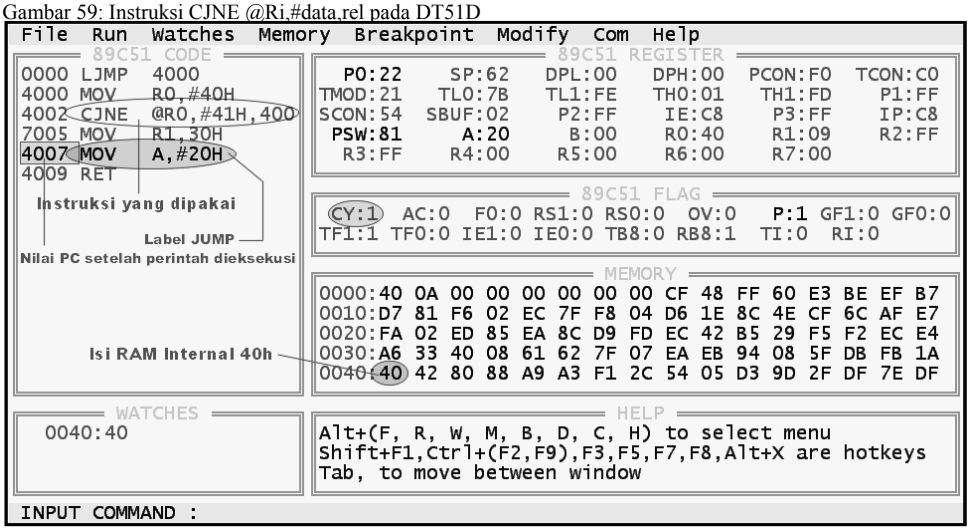
CJNE @R0,#41h,JUMP

akan mengecek apakah isi dari register R0 sama atau tidak dengan immediate data yang ada, apabila tidak sama akan melompat ke label JUMP yang berada di lokasi 4007h. Lihat Gambar 59.

Catatan: Sebelum memodifikasi isi memori, tentukan range memori yang akan dimodifikasi kemudian tentukan alamat memori yang akan dimodifikasi. Setelah itu isikan data yang dikehendaki. Dalam contoh diatas alamat memorinya adalah alamat 0040h dan datanya adalah 40h.

Untuk memonitor isi dari RAM internal alamat 40h digunakan fasilitas Watches yang tersedia pada DT51D. Tentukan alamat memori yang akan diwatch. Dalam contoh diatas alamat memorinya adalah alamat 0040h.

Gambar 59: Instruksi CJNE @Ri,#data,rel pada DT51D



DJNZ <byte>,<rel-addr>

Fungsi: Decrement, dan lompat jika hasilnya <> 0.

Deskripsi:

DJNZ berfungsi untuk meng-decrement suatu variabel byte. Jika hasil decrement ini bukan nol, maka program akan melompat ke alamat yang ditunjukkan oleh rel-addr. Variabel byte bernilai 00h akan underflow menjadi 0FFh. Tidak ada flag yang dipengaruhi perintah ini.

Pada saat perintah ini dieksekusi, nilai PC akan di-increment sampai diperoleh alamat dari perintah berikutnya yang mengikuti DJNZ. Setelah itu alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini.

Variabel byte yang dapat digunakan di sini adalah register atau direct byte.

Catatan: Ketika instruksi ini diterapkan pada suatu port, maka nilai yang digunakan adalah nilai dari latch output, dan bukannya nilai dari pin input.

Contoh:

Mula-mula RAM internal alamat 40h, 50h, dan 60h berturut-turut bernilai 01h, 70h, dan 15h. Perintah berikut ini :

DJNZ 40h,LABEL_1
DJNZ 50h,LABEL_2
DJNZ 60h,LABEL_3

menyebabkan program melompat ke LABEL_2, dan isi ketiga alamat RAM internal tersebut adalah 00h, 6Fh, dan 15h. Program tidak melompat ke LABEL_1 karena hasil DJNZ 40h menghasilkan bilangan 0. Lihat Gambar 61

DJNZ Rn,rel

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
(Rn) ← (Rn) - 1

IF (Rn) > 0 or (Rn) < 0
THEN
(PC) ← (PC) + rel

Contoh:

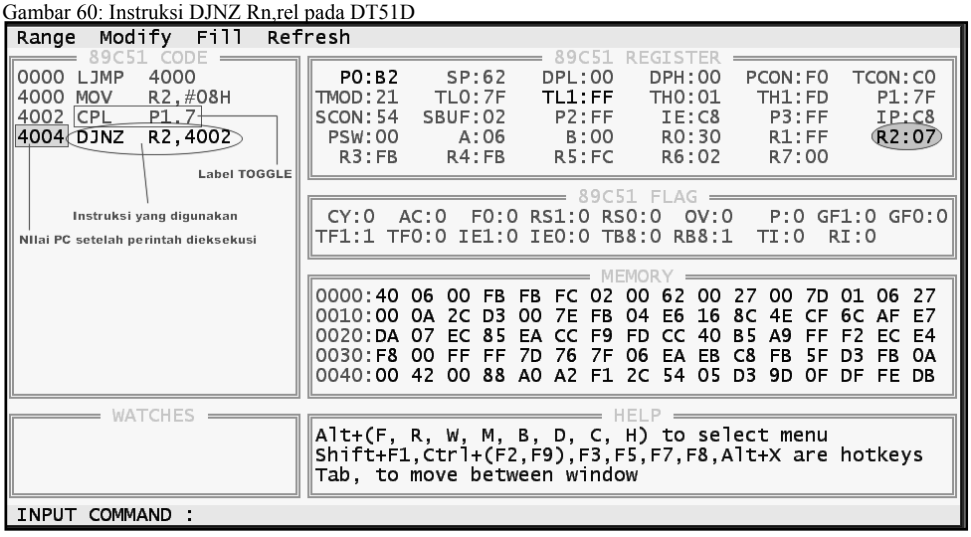
DJNZ seringkali digunakan dalam perulangan (*looping*). Instruksi berikut ini :

```
        MOV   R2, # 8h
TOGGLE: CPL   P1.7
        DJNZ  R2,TOGGLE
```

menyebabkan proses *looping* ke label TOGGLE akan terus berlangsung sampai isi dari Register R2 bernilai 00h. Lihat Gambar 60

Catatan: Gambar di bawah ini diambil pada saat perintah DJNZ R2,TOGGLE dieksekusi sekali.

Gambar 60: Instruksi DJNZ Rn,rel pada DT51D



DJNZ direct,rel

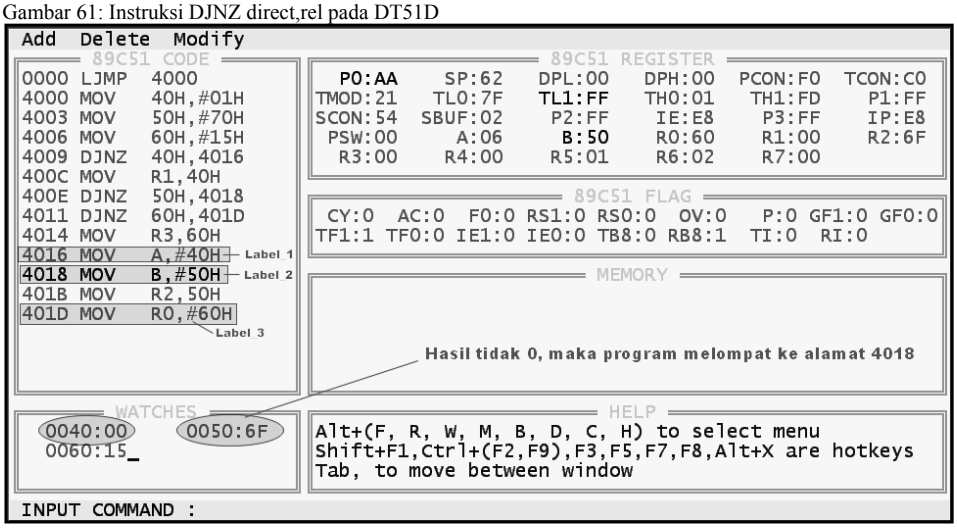
Jumlah byte: 3 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
(direct) ← (direct) – 1
IF (direct) > 0 or (direct) < 0
THEN
(PC) ← (PC) + rel

Contoh:

*lihat **DJNZ <byte>,<rel-addr>***

Gambar 61: Instruksi DJNZ direct,rel pada DT51D



JMP @A+DPTR

Fungsi: Jump indirect.

Deskripsi:

JMP @A+DPTR akan menjumlahkan nilai Akumulator dan DPTR (keduanya dianggap sebagai bilangan unsigned-integer), dan meletakkan hasilnya di Program Counter (PC). Baik nilai Akumulator maupun DPTR tidak berubah setelah eksekusi perintah. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Akumulator berisi suatu bilangan genap dari 0 s.d. 6 (0, 2, 4, 6). Perintah di bawah ini akan menyebabkan program melompat ke salah satu label (LABEL0 s.d. LABEL3), bergantung pada nilai Akumulator.

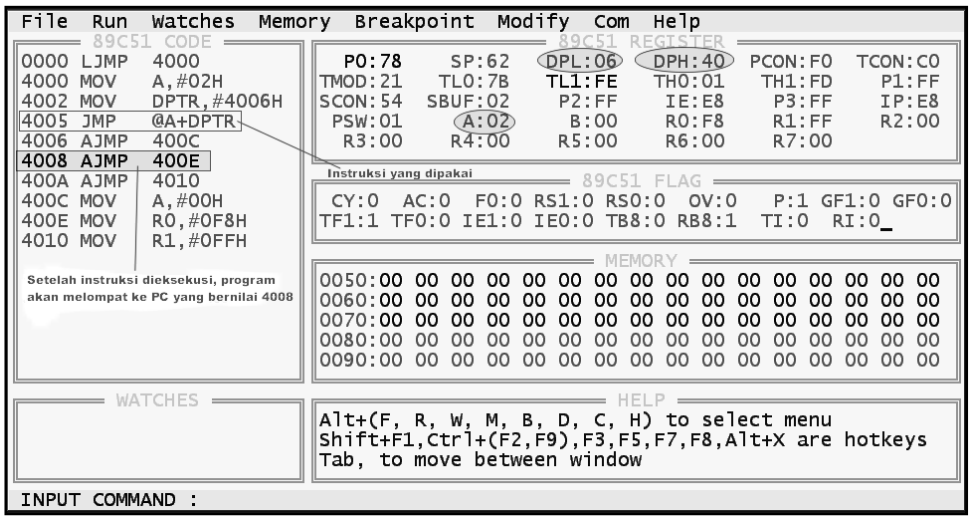
```
                MOV DPTR, # TBL
                JMP @A+DPTR
TBL:            AJMP    LABEL1
                AJMP    LABEL2
                AJMP    LABEL3
LABEL1:         MOV     A,#00H
LABEL2:         MOV     R0,#0F8H
LABEL3:         MOV     R1,#0FFH
```

Sebagai contoh, jika Akumulator bernilai 02h, maka program akan melompat ke alamat yang ditunjukkan oleh LABEL2. Lihat Gambar 62

Jumlah byte: 1 Jumlah cycle: 2

Operasi: (PC) ← (A) + (DPTR)

Gambar 62: Instruksi JMP @A+DPTR pada DT51D



JNZ rel

Fungsi: Lompat jika Akumulator <> 0

Deskripsi:

Jika nilai Akumulator <> 0, maka program akan melompat ke alamat yang disebutkan; jika tidak program akan melanjutkan ke baris berikutnya (tidak terjadi pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Baik Akumulator maupun flag-flag tidak ada yang dipengaruhi oleh perintah ini.

Contoh:

Akumulator mula-mula bernilai 00H. Perintah berikut ini :

```
    JNZ LABEL1
    INC A
```

JNZ LABEL2

menyebabkan Akumulator bernilai 01h dan program melompat ke alamat yang ditunjukkan oleh LABEL2.

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
IF (A) ≠ 0
THEN
(PC) ← (PC) + rel

JZ rel

Fungsi: Lompat jika Akumulator = 0

Deskripsi:

Jika Akumulator = 0, maka program akan melompat ke alamat yang disebutkan; jika tidak program akan melanjutkan ke baris berikutnya (tidak terjadi pelompatan). Ketika instruksi ini dieksekusi, maka nilai PC akan di-increment sampai diperoleh alamat perintah berikutnya yang mengikuti JB. Selanjutnya alamat tujuan dihitung dengan cara menambahkan offset rel (-128 s.d. +127) dengan nilai PC yang telah di-increment ini. Baik Akumulator maupun flag-flag tidak ada yang dipengaruhi oleh perintah ini.

Contoh:

Akumulator mula-mula bernilai 01h. Perintah berikut ini :

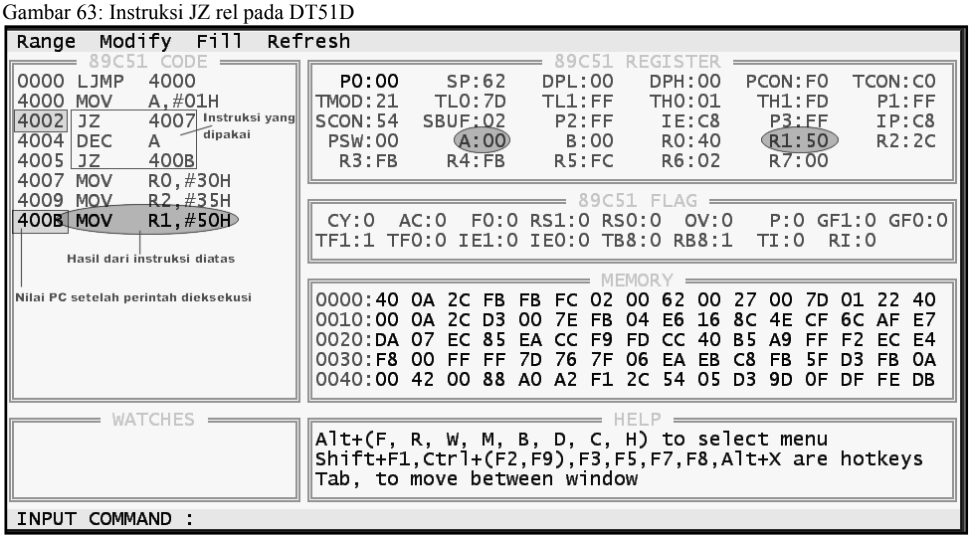
JZ LABEL1
DEC A
JZ LABEL2

Menyebabkan Akumulator bernilai 00h dan program melompat ke alamat yang ditunjukkan oleh LABEL2 yang berada di alamat 400Bh. Lihat Gambar 63

Jumlah byte: 2 Jumlah cycle: 2

Operasi: (PC) ← (PC) + 2
IF (A) = 0
THEN
(PC) ← (PC) + rel

Gambar 63: Instruksi JZ rel pada DT51D



RET

Fungsi: Kembali dari suatu sub-rutin.

Deskripsi:

RET berfungsi untuk mem-pop high-byte dan low-byte dari stack ke PC (secara berurutan), dan men-decrement Stack Pointer 2 kali. Tidak ada flag yang terpengaruh oleh perintah ini.

Contoh:

Mula-mula Stack Pointer berisi data 64h, perintah berikut:

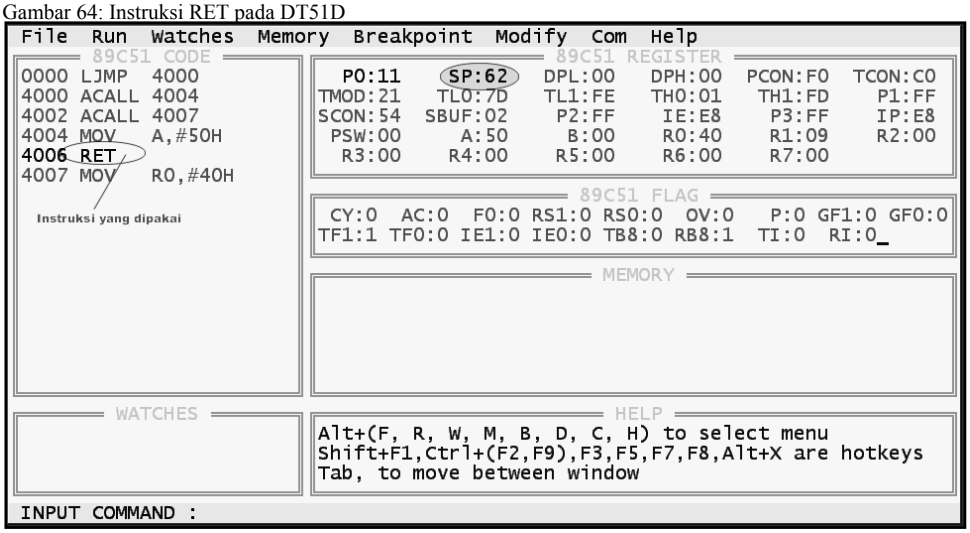
```
                ACALL START
                ACALL INT
START: MOV      A,#50H
                RET
INT:            MOV     R0,#40H
```

menyebabkan Stack Pointer bernilai 62h. Selanjutnya eksekusi program akan dilanjutkan dari alamat 4002h. Lihat Gambar 64

Jumlah byte: 1 Jumlah cycle: 2

Operasi: $(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) – 1$

Gambar 64: Instruksi RET pada DT51D



RETI

Fungsi: Kembali dari interrupt.

Deskripsi:

RETI berfungsi untuk mem-pop high-byte dan low-byte dari stack ke PC (secara berurutan) dan mengembalikan interrupt ke kondisi semula seperti sebelum terjadinya interrupt. Selain itu Stack Pointer di-decrement 2 kali, dan tidak ada flag yang terpengaruh oleh perintah ini.

Kondisi PSW (Program Status Word) tidak dikembalikan ke kondisi mula-mula sebelum terjadinya interrupt. Karena itu user harus menyelamatkan sendiri nilai PSW mula-mula ini, misalnya dengan PUSH PSW pada awal interrupt-handler.

Jika pada saat interrupt-handler diproses terdapat interrupt lain yang priority levelnya sama atau lebih rendah, maka interrupt yang terakhir ini akan di-pending. Ketika RETI sudah dieksekusi, maka program akan menjalankan 1 perintah terlebih dahulu sebelum akhirnya memproses interrupt yang sedang di-pending tersebut.

Contoh:
Mula-mula Stack Pointer berisi data 20h. Perintah berikut :

```
        ORG   4003H
        LJMP  INT
        ORG   4000H
        LJMP  START
INT:    NOP
        RETI
START:  MOV   SP,#20H
        ORL   IE,#81H
        NOP
        SETB  IE.1
        SETB  IE.0
        NOP
```

menyebabkan isi dari register IE yang semula bernilai 0E9h berubah menjadi 0EBh.

Jumlah byte: 1 Jumlah cycle: 2
Operasi: $(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) – 1$

NOP

Fungsi: Tidak melakukan apa-apa.
Deskripsi:
Ketika perintah ini dijumpai, mikrokontroler tidak melakukan apa-apa. Selain PC, tidak ada register dan flag lain yang terpengaruh oleh perintah ini.
Contoh:
Diinginkan supaya P2.7 mengeluarkan suatu pulsa active-low dengan lebar pulsa 5 cycle. Hal tersebut dapat diperoleh dari perintah berikut ini :

```
CLR   P2.7   ; Start
NOP          ; 1 Cycle
NOP          ; 1 Cycle
NOP          ; 1 Cycle
NOP          ; 1 Cycle
SETB  P2.7   ; 1 Cycle
```

Lihat Gambar 66

Jumlah byte: 1 Jumlah cycle: 1
Operasi: (PC) ← (PC) + 1

Gambar 66: Instruksi NOP pada DT51D