#### **TUGAS 2**

**SET INSTRUKSI UNTUK PROSESSOR INTEL DAN ARM (ANDROID)**



**NAMA : I Gusti Banjar Jawi**

**NIM : L200130173**

**KELAS : A**

**Program Studi Informatika**

**Fakultas Komunikasi dan Informatika**

**Universitas Muhammadiyah Surakarta**

1. **Set Instruksi (Instruction Set)**

Set Instruksi (Instruction Set atau Instruction Set Architecture (ISA)) didefinisikan sebagai suatu aspek dalam arsitektur komputer yang dapat dilihat oleh para pemrogram. Secara umum, ISA ini mencakup jenis data yang didukung, jenis instruksi yang dipakai, jenis register, mode pengalamatan, arsitektur memori, penanganan interupsi, eksepsi, dan operasi I/O eksternalnya (jika ada). ISA merupakan sebuah spesifikasi dari kumpulan semua kode-kode biner (opcode) yang diimplementasikan dalam bentuk aslinya (native form) dalam sebuah desain prosesor tertentu. Kumpulan opcode tersebut, umumnya disebut sebagai bahasa mesin (machine language) untuk ISA yang bersangkutan. ISA yang populer digunakan adalah set instruksi untuk chip Intel x86, IA-64, IBM PowerPC, Motorola 68000, Sun SPARC, DEC Alpha, dan lain-lain.  
ISA kadang-kadang digunakan untuk membedakan kumpulan karakteristik yang disebut di atas dengan mikroarsitektur prosesor, yang merupakan kumpulan teknik desain prosesor untuk mengimplementasikan set instruksi (mencakup microcode, pipeline, sistem cache, manajemen daya, dan lainnya).

### Daftar Set Instruksi 8051 dan Contoh Program untuk Setiap Kelompok Instruksi (Addressing Mode)

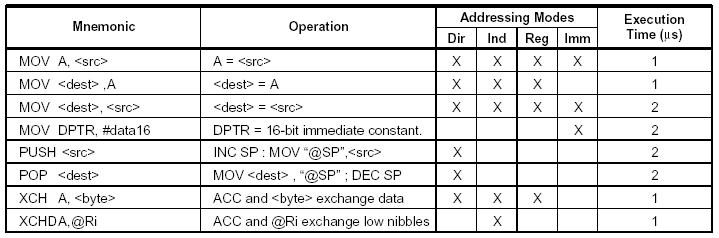
* **Set intruksi 8051 dibagi menjadi 3 kelompok besar, yaitu :**

1. Intruksi-intruksi transfer data

2. Intruksi-intruksi pemrosesan data

3. Intruksi-intruksi lompatan

1. **Intruksi-intruksi transfer data**
   * Daftar Intruksi Transfer data

[](http://blog.ub.ac.id/rusliawan/files/2012/03/Instruksi-instruksi-Transfer-Data.jpg)

* 1. **Instruksi Transfer Data**
     + RAM Internal

Perintah perpindahan data (MOV, XCH, POP, PUSH) pada RAM internal membutuhkan 1 sampai 2 cycle. Format instruksi :

MOV (tujuan), (asal)

Memungkinkan data untuk berpindah diantara 2 lokasi RAM internal atau SFR tanpa harus melalui akumulator terlebih dahulu.

* + RAM Ekternal

Perintah MOV 16-bit digunakan untuk inisialisasi DPTR atau untuk akses data 16-bit pada memori ekternal.perpindahan data antera memori internal dan ekternal menggunakan indirect addressing dengan menggunakan alamat 1-byte (@R1) atau 2-byte (@DPTR).

* + Look Up Tables

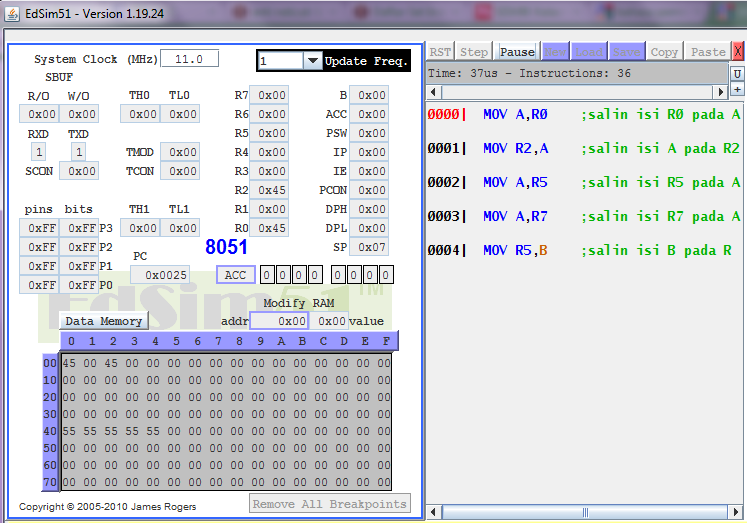
Ada dua perintah untuk membaca look-up tables pada ROM. MOVC (move constant) menggunakan program counter sebagai base register dan akumulator sebagai offsetnya.

MOVC A, @A+DPTR

Perintah tersebut dapat mengakses 256 entri. Nomor entri dimasukkan ke akumulator dan awal tabelnya pada DPTR.

MOVC A, @A+PC

**Berikut Contoh Program EdSim 51 untuk transfer data :**

[](http://blog.ub.ac.id/rusliawan/files/2012/03/move.png)

1. **Instruksi-intruksi Pemrosesan Data**

Instruksi-intruksi pemrosesan data dibagi manjadi 2 :

- Instruksi-instruksi Aritmatika

- Instruksi-intruksi Logika

1. **Intruksi-intruksi Aritmatika**

Set intruksi Aritmatika terdiri dari :

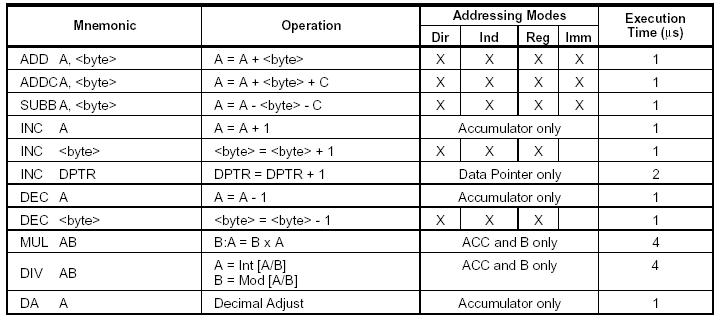
- Instruksi-instruksi penjumlahan dan pengurangan

- Instruksi-instruksi perkalian dan pembagian

- Instruksi-instruksi Increment dan Decrement

- Instruksi pengubahan ke decimal (Decimal Adjust)

**Daftar Set intruksi**

[](http://blog.ub.ac.id/rusliawan/files/2012/03/Instruksi-instruksi-Aritmatika.jpg)

Dengan beberapa macam type addresing maka instruksi aritmatika dapat dituliskan dengan setiap macam addresing seperti :

ADD A, 7FH (direct addressing)

ADD A, @R0 (indirect addressing)

ADD A, R7 (register addressing)

ADD A, #35H (immediate addressing)

Semua instruksi aritmatika mempunyai waktu eksekusi 1 cycle kecuali INC DPTR (2 cycle),MUL AB dan DIV AB (4 cycle). Perintah INC dapat berperasi pada data pointer 16-bit, yang biasanya digunakan untuk alamat 16-bit pada memori aksternal. Karena perintah DEC untuk data pointer ada, maka diperlukan serangkaian perintah sebagai berikut :

DEL DPL    ; pengurangan low-byte dari DPTR

MOV R7, DPL  ; disalin ke R7

CJNE R7, #0FFH, SKIP  ; pindah jika R7 dibawahb 0FFH

DEC DPH  ; mengurangi high-byte

SKIP   ; melanjutkan program

2.2 Instruksi-instruksi Logika

set instruksi logika terdiri dari :

- Instruksi-instruksi AND, OR, dan EX-OR

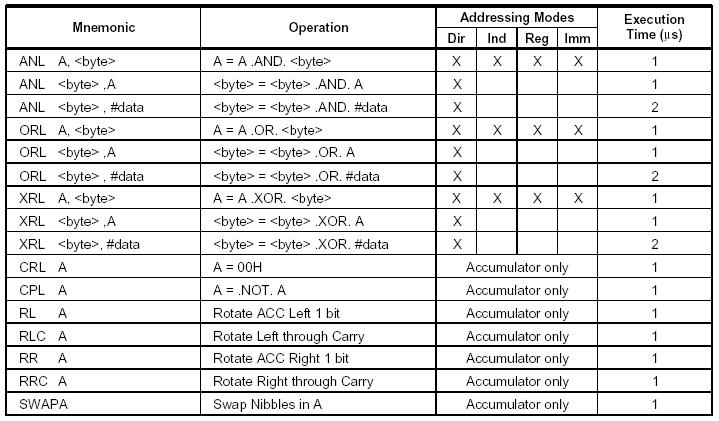
- Instruksi-instruksi komplemen dan clear

- Instruksi-instruksi putar

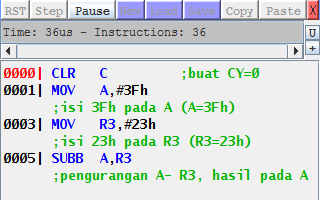
- Instruksi SWAP

- Instruksi logika per-bit

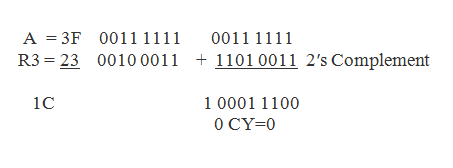
**Daftar set intruksi**

[](http://blog.ub.ac.id/rusliawan/files/2012/03/Instruksi-instruksi-Logika.jpg)

**Berikut contoh program EdSim 51 untuk pemrosesan intruksi aritmatika:**

[](http://blog.ub.ac.id/rusliawan/files/2012/03/aritmatika.png)

Prosesnya :

[](http://blog.ub.ac.id/rusliawan/files/2012/03/ada.png)

CY=0, AC=0, dan kita harus memeriksa kondisi Carry tersebut untuk mengetahui hasilnya positif atau negatif.

1. **Instruksi-Instruksi Lompatan**

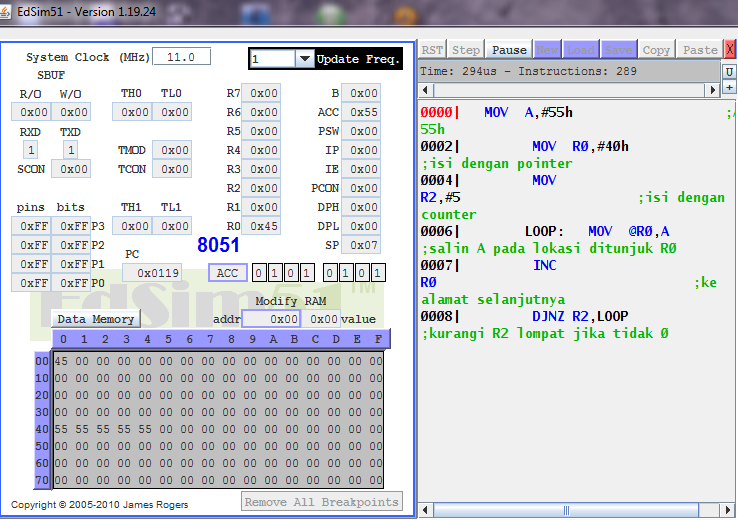
Instruksi-instruksi lompatan dibagi menjadi 3 yaitu:

Instruksi-instruksi lompatan tak bersyarat

Instruksi-instruksi lompatan bersyarat

Instruksi CALL dan RET

**Berikut Contoh program untuk menyalin 55h ke dalam memory lokasi 40 s/d 15h menggunakan intruksi lompatan Loop menggunakan edsim 51 :**

[](http://blog.ub.ac.id/rusliawan/files/2012/03/acc.png)

**berikut ini rincianya :**

        MOV  A,#55h                 ;A= 55h

        MOV  R0,#40h               ;isi dengan pointer

        MOV  R2,#5                   ;isi dengan counter

         LOOP:   MOV  @R0,A  ;salin A pada lokasi ditunjuk R0

        INC  R0                          ;ke alamat selanjutnya

        DJNZ R2,LOOP              ;kurangi R2 lompat jika tidak 0

Beberapa Jenis instruksi dan fungsinya secara lebih jelas diuraikan dalam tabel di bawah ini:

|  |  |  |
| --- | --- | --- |
| ACALL | Absolute Call | Memanggil subrutin program |
| ADD | Add Accumulator | Instruksi ADD digunakan untuk melakukan penambahan pada dua buah operand. Dan destination (tempat hasil dari proses) selalu pada A, sdang operand source dapat berupa register, data langsung, maupun memory |
| ADDC | Add Accumulator (With Carry) | Instruksi ADD digunakan untuk melakukan penambahan pada dua buah operand dengan carry |
| AJMP | Absolute Jump | AJMP ini adalah lompat tidak bersyarat jarak menengah. Disebut juga sebagai Jump 11-bit. Ini adalah instruksi 2-byte. Menjangkau alamat instruksi tepat di bawah AJMP, dan alamat label yang dituju, harus berada pada blok 2 KB yang sama. |
| ANL | AND Logic | Instruksi ini adalah melakukan AND logika pada dua operand dan menaruh hasilnya pada destination (Akumulator) |  |
| CJNE | Compare and Jump if Not Equal | Membandingkan data langsung dengan lokasi memori yang dialamati oleh register atau Akumulator jika tidak sama maka instruksi akan menuju ke alamat kode |  |
| CLR | Clear Register | Mereset isi register |  |
| CPL | Complement Register | Mengkomplement isi register |  |
| DA | Decimal Adjust | Mengkoreksi masalah yang timbul yang berkaitan denga penjumlahan bilangan BCD |  |
| DEC | Decrement Register | Mengurangi isi lokasi memori yang ditujukan oleh register R dengan 1, dan hasilnya disimpan pada lokasi tersebut |  |
| DIV | Divide Accumulator | Melakukan operasi pembagian |  |
| DJNZ | Decrement Register and Jump if Not Zero | Mengurangi nilai register dengan 1 dan jika hasilnya sudah 0 maka instruksi selanjutnya akan dieksekusi. Jika belum 0 akan menuju ke alamat kode |  |
| INC | Increment Register | Menambahkan isi memori dengan 1 dan menyimpannya pada alamat tersebut |  |
| JB | Jump if Bit Set | Membaca data per satu bit, jika data tersebut adalah 1 maka akan menuju ke alamat kode dan jika 0 tidak akan menuju ke alamat kode. |  |
| JBC | Jump if Bit Set and Clear Bit | Membaca data per satu bit, jika data tersebut adalah 1, selain akan melompat ke instruksi lain juga akan menolkan bit yang baru saja diperiksa. |  |
| JC | Jump if Carry Set | Membaca data carry |  |
| JMP | Jump to Address | Instruksi untuk memerintahkan menjangkau ke alamat kode tertentu |  |
| JNB | Jump if Bit Not Set | Membaca data per satu bit, jika data tersebut adalah 0 maka akan menuju ke alamat kode dan jika 1 tidak akan menuju ke alamat kode. |  |
| JNC | Jump if Carry Not Set (jump if no carry, jump if CY=1) | Instruksi ini, menggunakan carry sebagai menentu keputusan dalam jump. Jika CY=1, maka program akan melompat ke alamat yang ditunjuk. Namun jika CY=0, maka program akan mengeksekusi instruksi selanjutnya dibawah JNC tersebut. |  |
| JNZ | Jump if Accumulator Not Zero | Instruksi ini tidak akan memeriksa isi A. Jika 00, maka program akan melompat ke alamat yang ditunjuk |  |
| JZ | Jump if Accumulator Zero | JZ (lompat jika A=0), atau JC (lompat jika CY=1), akan membuat program melompat pada lokasi yang ditunjuk hanya jika kondisi yang diminta terpenuhi |  |
| LCALL | Long Call | Ini adalah instruksi 3-byte. Byte pertama adalah opcode, sedang 2-byte lainnya adalah alamat 16-bit yang dituju. Saat instruksi LCALL ini dijalankan, CPU tidak lagi mengeksekusi instruksi-instruksi di bawah LCALL, namun segera melompat pada alamat yang dituju |  |
| LJMP | Long Jump | Instruksi 3-byte, di mana byte pertama adalah opcode, sedang dua byte yang lain adalah representasi dari alamat 16-bit yang dituju |  |
| MOV | Move Memory | Instruksi ini untuk memindahkan isi akumulator/register atau data dari nilai luar atau alamat lain |  |
| MOVC | Move Code Memory | Membedakan bahwa instruksi ini dipakai di memori program |  |
| MOVX | Move Extended Memory | Perintah yang dipakai untuk memori data eksternal |  |
| MUL | Multiply | Melakukan operasi perkalian |  |
| NOP | No Operation | Menyisipkan instruksi untuk tidak mengerjakan apa-apa |  |
| ORL | OR Logic | Operand tujuan dan sumber di-OR-kan, dan menempatkan hasilnya pada tujuan destination. Instruksi ORL dapat digunakan untuk men-Set menjadi 1′s beberapa bit dalam register. |  |
| POP | Pop Value From Stack | Memanggil subrutin dengan instruksi CALL, memory stack akan menyimpan alamat di mana CPU akan kembali setelah menjalankan subrutin |  |
| PUSH | Push Value Onto Stack | Memanggil subrutin dengan instruksi CALL, memory stack akan menyimpan alamat di mana CPU akan kembali setelah menjalankan subrutin |  |
| RET | Return From Subroutine | Intruksi untuk kembali dari suatu subrutin program ke alamat terakhir subrutin tersebut di panggil |  |
| RETI | Return From Interrupt | Mereset bit yang bersangkutan |  |
| RL | Rotate Accumulator Left | Pada putar kiri , 8-bit dalam akumulator digeser ke kiri sejauh satu bit. Bit D7 keluar dari Most Significant Bit (MSB) dan ditempatkan pada D0 atau Least Significant Bit (LSB) |  |
| RLC | Rotate Accumulator Left Through Carry | Menggeser 8-bit dalam akumulator ke kanan sejauh 1 bit melewati Carry. Bit D0 keluar dari Least Significant Bit (LSB) dan ditempatkan pada Carry. Sedangkan isi Carry ditempatkan pada D7 atau Most Significant Bit (MSB) |  |
| RR | Rotate Accumulator Right | Pada putar kanan , 8-bit dalam akumulator digeser ke kanan sejauh satu bit. Bit D0 keluar dari Least Significant Bit (LSB) dan ditempatkan pada D7 atau Most Significant Bit (MSB) |  |
| RRC | Rotate Accumulator Right Through Carry | Menggeser 8-bit dalam akumulator ke kanan sejauh 1 bit melewati Carry. Bit D0 keluar dari Least Significant Bit (LSB) dan ditempatkan pada Carry. Sementara isi Carry ditempatkan pada D7 atau Most Significant Bit (MSB) |  |
| SETB | Set Bit | Instruksi untuk mengaktikan atau memberikan logika 1 pada sebuat bit data |  |
| SJMP | Short Jump | SJMP adalah lompat tanpa syarat jarak pendek. Disebut juga sebagai Jump relatif 8-bit. Ini adalah instruksi 2 byte. Byte pertama adalah opcode, sedang byte lainnya adalah alamat relatif yang dituju. Ya alamat relatif, sehingga nilai pada byte ke dua ini bukan representasi dari alamat yang dituju, melainkan nilai relatif terhadap nilai PC saat itu |  |
| SUBB | Subtract From Accumulator With Borrow | SUBB bisa difungsikan sebagai SUB. Yaitu dengan membuat/memastikan CY=0 sebelum perintah SUBB dilaksanakan |  |
| SWAP | Swap Accumulator Nibbles | men-swap (menukar balik) nible bawah dan nible atas |  |
| XCH | Exchange Bytes | Merubah bit |  |
| XCHD | Exchange Digits | Merubah digit |  |
| XRL | Exclusive OR Logic | Membalik nilai (complement) beberapa bit tertentu dari sebuah bilangan biner 8 bit, caranya dengan membentuk sebuah bilangan biner 8 bit sebagai data konstan yang di-XRL-kan bilangan asal. Bit yang ingin dibalik-nilai diwakili dengan ‘1’ pada data konstan, sedangkan bit lainnya diberi nilai ‘0’ |  |

* **Set Instruksi Pada Microprocessor 8088**

1. Transfer Data

Set instruksi 8088 mencakup 14 transfer data yang menggerakkan byte atau kata data antara memori dan register selain juga antara akumulator dan port I/O.

1. GeneralPurpose(Umum) : MOV,PUSH,POP,XCHG,SXLAT
2. Input/Output : In, Out
3. Address Transfer (Pemindahan Alamat) : LEA, KDS, KES
4. Flag Transfer (Pemindahan Flag) : LAHF, SAHF, PUSHF, POPF
5. Arithmetic Instructions (Instruksi Perhitungan)

8088 mampu melakukan penambahan, pengurangan, pengurangan dan pembagian data selain juga bytes dan kata. Sistem ini menambah dan mengurangi dengan menggunakan bytes atau kata yang bertanda atau tak bertanda dan data BCD atau ASCII.

1. Addition (Penjumlahan) : ADD, ADC, AAA, DAA, INC
2. Substraction (Pengurangan) : SUB, SBB, AAS, DAS, DES, NEG, CMP
3. Multiplication (Perkalian) : MUL, IMUL, AAM
4. Division (Pembagian) : DIV, IDIV, AAD
5. Sign Extension : CBW, CWD
6. Manipulasi Bit

Instruksi ini mencakup operasi logika, shift dan rotasi.

1. Logical (Logika) : AND, OR, XOR, NOT, TEST
2. Shift (Geser) : SAL/SHL, SAR, SHR
3. Rotate (Gulung) : ROL, ROR, RCL, RCR
4. Instruksi String

Instruksi string digunakan untuk memanipulasi string data dalam memori. Setiap string tersusun baik dari bytes maupun kata dan hingga mencapai 64 K bytes panjang.nya

1. Direction : CLD, STD
2. Repeat Prefixes : REP, REPE/REPZ, REPNE/REPNZ,
3. Move : MOVSB, MOVSW
4. Compare : CMPS, CMPSB, CMPSW
5. Scan : CSAS, SCASB, SCASW
6. Load and Store : LODS, LODSB, LODSW, STOS, STOSB, STOSW
7. Transfer Program

Instruksi transfer program mencakup jump,  call, dan return, instruksi yang sudah kita kenal dalam 8085.

1. Unconditional transfer (Pemindahan)               : CAL, RET, JMP
2. Conditional transfer (Pemindahan Bersyarat)   : JA/JNBE, JAE/JNB, JB/JNAE/JC, JBE/JNA, JCXZ, JE/JZ, JG/JNLE, JGF/JNL, JL/JNGE, JLE/JNG, JNC, JNE/JNZ, JNO, JNP/JPO, JNS, JO, JP/JPE, JS
3. Loop/Jump                                                    : LOOP, LOOPE/LOOPZ, LOOPNE/LOOPNZ
4. Kontrol Prosesor

Instruksi kontrol prosesor memungkinkan dan tidak memungkinkan interupsi, memodifikasi bit flag dan mensinkronisasi kejadian eksternal.

1. Flag Operations                            : CLC, STC, CMC, CLD, STD, CLI, STI,
2. External Synchronization Interrupt : INT, INTO, IRET

* Berikut ini penjelasan tentang instruksi yang digunakan pada bahasa rakitan 8086 dan 8088 atau sejenisnya :

1. **AAA** (ASCII adjust for edition) Pengaturan ASCII bagi penambahan
2. **AAD**(ASCII adjust for division) Pengaturan ASCII bagi pembagian
3. **AAM** (ASCII adjust for multipy) Pengaturan ASCII bagi perkalian
4. **AAS**(ASCII adjust for substraction) Pengaturan ASCII bagi pengurangan
5. **ADC** (add with carry) T ambahkan dengan carry
6. **ADD** (addition) Penambahan
7. **AND**(logic AND) Logik AND
8. **CALL**(CALL subroutine) Subrutin panggil
9. **CBW** (Convert byte to word) Konversikan byte ke kata
10. **CLC**(Clear Carry) Kosongkan corry flag
11. **CLD**(Clear directon flag) Kosongkan flag arah
12. **CLI** (Clear interrupt enable) Kosongkan flag penggerak interupsi
13. **DAS** (decimal adjust for substraction) Pengaturan decimal bagi pengurangan
14. **DEC** (decrement) Penurunan operand tujuan dengan 1
15. **DIV**(divide) Pembagian tak bertanda
16. **ESC** (escape) Escape digunakan sehubungan dengan suatu co-procesor external
17. **HLT**(halt) HLT menghentikan prosesor sampai saluran reset diaktifkan
18. **IDIV** (Integer division) Pembagian kilat
19. **IMUL** (Integer multiply) Perkalian kilat
20. **IN** (Input) IN (masukan) mentransfer data dari port yang disperifikasikan ke dalam register AC atau AX
21. **NC** (Increment) INC menaikkan operand tujuan dalam 1
22. **INT** (Interupt) INT (Interupsi) mengawali suatu prosedur interupsi dengan jenis yang dispesifikasikan oleh instruksi yang bersangkutan
23. **INTO** (Interupt on overflow) Instrupsi bila ada overflow INTO digunakan untuk membangkitkan suatu interupsi perangkat lunak yang bergantumg pada status flag OF
24. **IRET** (Interupt return) IRET digunakan untuk kembali dari suatu interupsi dan mendapatkan kembali IP,CS dan flag-flag dari stack
25. **JA**(Jump on above) Digunakan jika operand pertama lebih besar daripada operand kedua
26. **JNBE**(Jump) Digunakan jika pada saat pembandingan operand 1, tidak lebih kecil atausama dengan operand 2
27. **JAE** (Jump on above or equal) Digunakan pada saat pembandingan operand 1 lebih besar atau sama dengan operand 2
28. **JNB** (Jump not below) Digunakan pada saat pembandingan operand 1 tidak lebih kecil dari operand 2
29. **JB** (Jump on below) Digunakan pada saat operand 1 lebih kecil dari operand 2
30. **JNAE** (Jump on not above or equal) Digunakan pada saat operand 1 tidak lebih besar atau sama dengan operand 2
31. **JBE** (Jump on below or equal) Digunakan pada saat operand 1 lebih kecil atau sama dengan operand 2
32. **JNA** (Jump on not above) Digunakan pada saat operand 1 tidak lebih besar dari operand 2
33. **JC** (Jump on carry) Digunakan pada saat akan diprogram CF=1
34. **JCXZ** (Jump if CX=0) Digunakan bila isi register CX=0
35. **JE**(Jump on equal) Digunakan pada saat pembandingan kedua operand sama
36. **JZ** (Jump on zero) Digunakan pada saat akan diproses ZF=1
37. **JG** (Jump on greather than) Digunakan pada saat pembandingan operand 1 menunjukkan lebih besar dari operand 2
38. **JNLE** (Jump on greather or equal) Digunakan pada saat operand 1tidak lebih kecil atau sama dengan operand 2
39. **JGE** (Jump on greather or equal) Digunakan pada saat operand 1 lebih besar atau sama dengan operand 2
40. **JNL** (Jump on less) Digunakan pada saat operand 1 tidak lebih kecil dari operand 2
41. **JL** (Jump on less) Digunakan pada saat operand 1 lebih kecil dari operand 2
42. **JNGE** (Jump on not greather or equal) Digunakan pada saat operand 1 tidak lebih besar atau sama dengan operand 2
43. **JLE** (Jump on less or equal) Digunakan pada saat operand 1 lebih kecil atau sama dengan operand 2
44. **JNG** (Jump on not greather than) Digunakan pada saat operand 1 tidak lebih besar daripada operand 2
45. **JMP** (Unconditional jump) Lompatan tidak bersyarat
46. **JNC**(Jump on not carry) Digunakan pada saat diproses CF=O
47. **JNE** (Jump on not equal) Digunakan pada saat operand 1 tidak sama dari operand 2
48. **JNO** (Jump on not zero) Digunakan pada saat diproses ZF=0
49. **JNO** (Jump on not overflow) Digunakan bila tidak ada overflow (OF=O)
50. **JNS**(Jump on not sign) Digunakan pada saat diproses SF=O
51. **ZNP** (Jump on not pority) Digunakan bila tidak ada pointer
52. **JPO** (Jump on poity odd) Digunakan bila pointer ganjil
53. **O** (Jump on overflow) Digunakan pada saat ada overflow
54. **JP** (Jump on Pority equal) Digunakan bila pointer genap
55. **JS** (Jump on sign) Digunakan pada saat diproses ZF=1
56. **LAHF** (Load AH from flag) LAHF me-load bit 7,6,4,2 & 0 pada register AH masing-masing dengan isi flag SF.ZF,AF & CF
57. **LDS**(Load pointer using DS) Muatkan peminjak dengan menggunakan DS
58. **LEA** (Load effective address) LEA mentransfer opsan sumber 16 bit dalam memori ke tujuan 16 bit
59. **LES**(Load pointer using ES) Muatkan pointer dengan menggunakan ES
60. **LOCK** (Lock bas) Digunakan dalam penerapan pemakaian sumber bersama, untuk memastikan bahwa memori tidak diakses secara serentak oleh lebih dari satu proses
61. **LODS**(Load string) Muatkan string byte atau kata
62. **LOOP**Loop jika EX bukan 0
63. **LOOPE** dan LOOPZ -Loop bila sama
64. **-Loop**bila 0
65. **LOOPNE/LOOPNZ**-Loop bila tidak sama (ZF=0)
66. **-Loop**bila tidak nol (ZF=O)
67. **MOVE** Pindahkan byte atau kata memindahkan data 8 bit atau 16 bit
68. **MOUS** (Move string) Pindahkan data string 8 bit atau 16 bit
69. **MUL**(Multifly) Digunakan untuk mengalihkan isi bertandapada akumulator dengan suatu operand sumber yang dispesifikasikan
70. **NEG**(Negate) Mengurangkan suatu operand tujuan dari 0 & menyimpan hasil komplemen keduanya dalam tujuan
71. **NOP**(Logic NOT) NOP tidak membe rikan dampak tertentu (tidak ada operasi)
72. **NOT**(Logic OR) Operasi logic OR
73. **OR** (Logic OR) Operasi logic OR
74. **OUT** (Output) Digunakan untuk mentransfer data dari AL atau AX ke point (pangkalan)
75. **POP** (POP from stack) Keluaran data dari stack dari AL atau AX ke point (pangkalan)
76. **POPF**(POP flag) Keluarkan ke flag dan stack
77. **POSH** (Push to stack) Dorong sumber ke stack
78. **PUSHF**(Push flag) Dorong flag ke stack
79. **RCL** (Rotate lift with carry) Putar ke kirin dengan carry 1
80. **RCR** (Rotate right with carry) Putar ke kanan dengan carry 1
81. **REP**Ulangi
82. **REE**Ulangi jika sama
83. **REPZ** Ulangi jika nol
84. **REPNE** Ulangi jika tidak sama
85. **REPNZ** Ulangi jika tidak nol
86. **RET** (Return from subroutine) Digunakan untuk kembali ke subrutin
87. **ROL**(Rotate left) Rotasi (putar) ke kiri 1
88. **ROR** (Rotate right) Rotasi (putar) ke kanan 1
89. **SAHF** (store AH in flags register) Simpan AH dalam reegister flag
90. **SAL/SHL**(Shift left arithmetic/logical) Geser aritmatik/logika ke kiri 1
91. **SAR/SHR**(Shift right arithmetic/logical) Geser aritmatik/logika ke kanan 1
92. **SBB** (Substract with borrow) Kurangkan dengan borrow
93. **SCAS** (Scan string) Digunakan untuk mengurangkan string tujuan yang dialamatkan a/n regiser DI,dari AX atau AL
94. **STC** (Set carry flag) Menset carry flag (CF)
95. **STD** (Set direction flag) Menset flag arah (DF)
96. **STI** (Set interrupt enable flag) Menset flag penggerak interupsi (IF)
97. **STOS**(Substract) Memindahkan operand sumber yang terkandung dalam AX atau AL ke suatu tujuan yang dialamatkan oleh DI
98. **SUB** (Substract) Mengurangkan sumber dari operand tujuan dan hasilnya ditepatkan dalam tujuan
99. **TEST**(Logical comporison) Menguji operand-operand atau logic perbandingan
100. **WAIT** Menunggu sampai saluran test aktif
101. **XCHG** (Exchange) Pertukaran dan sumber
102. **XLAT** (Translate) Digunakan untuk menerjemahkan se-byte dalam register AL ke dalam suatu byte yang diambil dari tabel terjemahan
103. **XOR** (Logic exclusive OR) Operasi exclusive OR
104. **Set Instruksi ARM**

* Advanced RISC Machines (ARM) limited telah mendesain suatu famili mikroprosesor dan melisensikan desain tersebut ke perusahaan lain untuk fabrikasi chip yang penggunaannya dalam produk komputer dan sistem uang embedded.
* Perusahan ARM yang relatif baru, merupakan perkembangan dari perusahan Acorn Computer yang mengembangkan desain prosesor pada awal tahun 1980-an.
* Penggunaan utama mikroprosesor ARM adalah pada aplikasi embedded yang berdaya rendah dan berbiaya rendah, seperti misalnya mobile telephone, modem komunikasi, sistem manejemen mesin mobil, dan hand-held

**Instruksi set ARM**

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Meaning** | **Earliest CPU / Comments** |
| [*ABS*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Absolute Value | Floating Point 1 |
| [*ACS*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Cosine | Floating Point 5 |
| [**ADC**](https://www.heyrick.co.uk/assembler/mov.html#adc) | Add with Carry | - |
| *ADC* | Thumb: Add with Carry | Thumb |
| [**ADD**](https://www.heyrick.co.uk/assembler/mov.html#add) | Add | - |
| *ADD* | Thumb: Add | Thumb |
| [*ADF*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Add | Floating Point 1 3 |
| [ADR](https://www.heyrick.co.uk/assembler/pseudo.html#adr) | Get address of object (within 4K) | This is an assembler pseudo-instruction |
| [ADRL](https://www.heyrick.co.uk/assembler/pseudo.html#adrl) | Get address of object (beyond 4K) | This is an assembler pseudo-instruction |
| [ALIGN](https://www.heyrick.co.uk/assembler/pseudo.html#align) | Set the program counter to the next word boundary | This is an assembler pseudo-instruction |
| [**AND**](https://www.heyrick.co.uk/assembler/mov.html#and) | Logical AND | - |
| *AND* | Thumb: Logical AND | Thumb |
| [ASL](https://www.heyrick.co.uk/assembler/shift.html#asl) | Arithmetic Shift Left | This is an option, not an instruction |
| [*ASN*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Sine | Floating Point 5 |
| [ASR](https://www.heyrick.co.uk/assembler/shift.html#asr) | Arithmetic Shift Right | This is an option, not an instruction;  available on Thumb. |
| [*ATN*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Arc Tangent | Floating Point 5 |
| [**B**](https://www.heyrick.co.uk/assembler/bl.html#b) | Branch | - |
| *B* | Thumb: Branch | Thumb |
| [**BIC**](https://www.heyrick.co.uk/assembler/mov.html#bic) | Bit Clear | - |
| *BIC* | Thumb: Bit Clear | Thumb |
| *BKPT* | Thumb: Breakpoint | Thumb |
| [**BL**](https://www.heyrick.co.uk/assembler/bl.html#bl) | Branch with Link | - |
| *BL* | Thumb: Long Branch with Link | Thumb |
| *BLX* | Thumb: Branch with Link and Exchange | Thumb |
| *BX* | Thumb: Branch and Exchange | Thumb |
| [**CDP**](https://www.heyrick.co.uk/assembler/coprocmnd.html#cdp) | Co-processor data operation | - |
| *CDP2* | CDP, *non-conditional* so more co-processor commands possible | ARMv5 |
| *CLZ* | Count Leading Zeros | ARMv5 |
| [*CMF*](https://www.heyrick.co.uk/assembler/fpops.html#cmf) | Compare floating point value | Floating Point 1 3 |
| [**CMN**](https://www.heyrick.co.uk/assembler/cmp.html#cmn) | Compare negated values | - |
| *CMN* | Thumb: Compare negated values | Thumb |
| [**CMP**](https://www.heyrick.co.uk/assembler/cmp.html#cmp) | Compare values | - |
| *CMP* | Thumb: Compare values | Thumb |
| [*CNF*](https://www.heyrick.co.uk/assembler/fpops.html#cnf) | Compare negated floating point values | Floating Point 1 |
| [*COS*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Cosine | Floating Point 5 |
| [DCx](https://www.heyrick.co.uk/assembler/pseudo.html#dcx) | Define byte (B), halfword (W), word (D), string (S), or floating point (F) value. Some assemblers allow DCFS, DCFD, etc for FP precision. | This is an assembler pseudo-instruction |
| [*DVF*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Divide | Floating Point 1 3 |
| [**EOR**](https://www.heyrick.co.uk/assembler/mov.html#eor) | Exclusive-OR two values | - |
| *EOR* | Thumb: Logical Exclusive-OR | Thumb |
| [EQUx](https://www.heyrick.co.uk/assembler/pseudo.html#equx) | Define byte (B), halfword (W), word (D), string (S), or floating point (F) value. Some assemblers allow EQUFS, EQUFD, etc for FP precision. | This is an assembler pseudo-instruction |
| [*EXP*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Exponent | Floating Point 5 |
| *FABS* | VFP: Absolute | VFP |
| *FADD* | VFP: Addition | VFP |
| *FCMP* | VFP: Compare | VFP |
| *FCVTDS* | VFP: Single to Double | VFP |
| *FCVTSD* | VFP: Double to Single | VFP |
| *FCPY* | VFP: Copy [*like MVF*] | VFP |
| *FDIV* | VFP: Division | VFP |
| [*FDV*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast Divide | Floating Point 1 |
| [*FIX*](https://www.heyrick.co.uk/assembler/fpops.html#fix) | Convert floating value to an integer | Floating Point 1 3 |
| *FLD* | VFP: Load VFP registers | VFP |
| *FLDMDB* | VFP: Load multiple VFP registers, decr. before | VFP |
| *FLDMIA* | VFP: Load multiple VFP registers, incr. after | VFP |
| [*FLT*](https://www.heyrick.co.uk/assembler/fpops.html#flt) | Convert integer to a floating value | Floating Point 1 3 |
| *FMAC* | VFP: Multiply with Accumulate | VFP |
| *FMDHR* | VFP: Transfer ARM register to upper half of Double | VFP |
| *FMDLR* | VFP: Transfer ARM register to lower half of Double | VFP |
| *FMRDH* | VFP: Transfer upper half of Double to ARM register | VFP |
| *FMRDL* | VFP: Transfer lower half of Double to ARM register | VFP |
| [*FML*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast multiply | Floating Point 1 |
| *FMSC* | VFP: Multiply with Negate and Accumulate | VFP |
| *FMRS* | VFP: Transfer Single to ARM register | VFP |
| *FMSR* | VFP: Transfer ARM register to Single | VFP |
| *FMUL* | VFP: Multiply | VFP |
| *FMRX* | VFP: Transfer VFP system register to ARM register | VFP |
| *FMSTAT* | VFP: Transfer FPSCR flags to CPSR | VFP |
| *FMXR* | VFP: Transfer ARM register to VFP system register | VFP |
| *FNEG* | VFP: Copy Negative [*like MVN*] | VFP |
| *FNMAC* | VFP: Multiply with Deduct | VFP |
| *FNMSC* | VFP: Multiply with Negate and Deduct | VFP |
| *FNMUL* | VFP: Negative Multiply | VFP |
| [*FRD*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Fast reverse divide | Floating Point 1 |
| *FSITO* | VFP: Signed Integer to Float | VFP |
| *FSQRT* | VFP: Square Root | VFP |
| *FST* | VFP: Save VFP registers | VFP |
| *FSTMDB* | VFP: Save multiple VFP registers, decr. before | VFP |
| *FSTMIA* | VFP: Save multiple VFP registers, incr. after | VFP |
| *FSUB* | VFP: Subtraction | VFP |
| *FTOSI* | VFP: Float to Signed Integer | VFP |
| *FTOUI* | VFP: Float to Unsigned Integer | VFP |
| *FUITO* | VFP: Unsigned Integer to Float | VFP |
| [**LDC**](https://www.heyrick.co.uk/assembler/coprocmnd.html#ldc) | Load from memory to co-processor | - |
| *LDC2* | LDC, *non-conditional* so more co-processor commands possible | ARMv5 |
| [*LDF*](https://www.heyrick.co.uk/assembler/fpops.html#ldf) | Load floating point value | Floating Point 1 3 |
| [**LDM**](https://www.heyrick.co.uk/assembler/str.html#ldm) | Load multiple registers | - |
| *LDMIA* | Thumb: Load multiple registers | Thumb |
| [**LDR**](https://www.heyrick.co.uk/assembler/str.html#ldr) | Load register (32 bit) | - |
| *LDR* | Thumb: Load register (32 bits?) | Thumb |
| [**LDRB**](https://www.heyrick.co.uk/assembler/str.html#ldr) | Load byte (8 bit) into register | - |
| *LDRB* | Thumb: Load byte (8 bit) into register | Thumb |
| **LDRH** | Load halfword (16 bit) into register | StrongARM |
| *LDRH* | Thumb: Load halfwit (boo!) into register | Thumb |
| **LDRSB** | Load signed byte (sign + 7 bit) into register | StrongARM |
| *LDRSB* | Thumb: Load signed byte (sign + 7 bit) into register | Thumb |
| **LDRSH** | Load signed halfword (sign + 15 bit) into register | StrongARM |
| *LDRSH* | Thumb: Load signed halfword (sign + 15 bit) into register | Thumb |
| [*LFM*](https://www.heyrick.co.uk/assembler/fpops.html#lfm) | Load multiple floating point values | Floating Point 1 |
| [*LGN*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Logarithm to base e | Floating Point 5 |
| [*LOG*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Logarithm to base 10 | Floating Point 5 |
| [LSL](https://www.heyrick.co.uk/assembler/shift.html#lsl) | Logical Shift Left | This is an option, not an instruction;  available on Thumb. |
| [LSR](https://www.heyrick.co.uk/assembler/shift.html#lsr) | Logical Shift Right | This is an option, not an instruction;  available on Thumb. |
| [**MCR**](https://www.heyrick.co.uk/assembler/coprocmnd.html#mcr) | Co-processor register transfer (ARM to co-processor) | - |
| *MCR2* | MCR, *non-conditional* so more co-processor commands possible | ARMv5 |
| *MCRR* | MCR, with two registers transferred at one time | ARMv5TE |
| [**MLA**](https://www.heyrick.co.uk/assembler/mul.html#mla) | Multiply with Accumulate | - |
| [*MNF*](https://www.heyrick.co.uk/assembler/fpops.html#mnf) | Move negated | Floating Point 1 |
| [**MOV**](https://www.heyrick.co.uk/assembler/mov.html#mov) | Move value/register into a register | - |
| *MOV* | Thumb: Move value/register into a register | Thumb |
| [**MRC**](https://www.heyrick.co.uk/assembler/coprocmnd.html#mrc) | Co-processor register transfer (co-processor to ARM) | - |
| *MRC2* | MRC, *non-conditional* so more co-processor commands possible | ARMv5 |
| *MRRC* | MRC, with two registers transferred at one time | ARMv5TE |
| [**MRS**](https://www.heyrick.co.uk/assembler/psr.html#32bit) | Move status flags to a register | ARM 6 |
| [**MSR**](https://www.heyrick.co.uk/assembler/psr.html#32bit) | Move contents of a register to the status flags | ARM 6 |
| [*MUF*](https://www.heyrick.co.uk/assembler/fpops.html) | Multiply | Floating Point 1 3 |
| [**MUL**](https://www.heyrick.co.uk/assembler/mul.html#binop) | Multiply | - |
| *MUL* | Thumb: Multiply | Thumb |
| [*MVF*](https://www.heyrick.co.uk/assembler/fpops.html#mvf) | Move value/float register into a float register | Floating Point 1 3 |
| [**MVN**](https://www.heyrick.co.uk/assembler/mov.html#mvn) | Move negated | - |
| *MVN* | Thumb: Move negated | Thumb |
| *NEG* | Thumb Negate | Thumb |
| *NOP* | Thumb: No Operation | Thumb |
| [*NRM*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Normalise | Floating Point 1 |
| [OPT](https://www.heyrick.co.uk/assembler/opt.html) | Select assembly options | This is an assembler pseudo-instruction |
| [**ORR**](https://www.heyrick.co.uk/assembler/mov.html#orr) | Logical OR | - |
| *ORR* | Thumb: Logical OR | Thumb |
| *PLD* | PreLoaD | ARMv5 |
| [*POL*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Polar Angle | Floating Point 5 |
| *POP* | Thumb: Pop registers from stack | Thumb |
| [*POW*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Power | Floating Point 5 |
| *PUSH* | Thumb: Push registers onto stack | Thumb |
| *QADD* | Add, saturating | ARMv5E |
| *QDADD* | Add, double saturating | ARMv5E |
| *QDSUB* | Subtract, double saturating | ARMv5E |
| *QSUB* | Subtact, saturating | ARMv5E |
| [*RDF*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Reverse Divide | Floating Point 1 |
| [*RFC*](https://www.heyrick.co.uk/assembler/fpops.html#rfc) | Read FP control register | Floating Point 1 4 |
| [*RFS*](https://www.heyrick.co.uk/assembler/fpops.html#rfs) | Read FP status register | Floating Point 1 3 |
| [*RMF*](https://www.heyrick.co.uk/assembler/fpops.html#rmf) | Remainder | Floating Point 2 3 |
| [*RND*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Round to integral value | Floating Point 2 3 |
| [ROR](https://www.heyrick.co.uk/assembler/shift.html#ror) | Rotate Right | This is an option, not an instruction;  available on Thumb. |
| [*RPW*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Reverse Power | Floating Point 5 |
| [RRX](https://www.heyrick.co.uk/assembler/shift.html#rrx) | Rotate Right with extend | This is an option, not an instruction |
| [**RSB**](https://www.heyrick.co.uk/assembler/mov.html#rsb) | Reverse Subtract | - |
| [**RSC**](https://www.heyrick.co.uk/assembler/mov.html#rsc) | Reverse Subtract with Carry | - |
| [*RSF*](https://www.heyrick.co.uk/assembler/fpops.html#rsf) | Reverse Subtract | Floating Point 1 |
| [**SBC**](https://www.heyrick.co.uk/assembler/mov.html#sbc) | Subtract with Carry | - |
| *SBC* | Thumb: Subtract with Carry | Thumb |
| [*SFM*](https://www.heyrick.co.uk/assembler/fpops.html#sfm) | Store Muliple Floating point values | Floating Point 1 |
| [*SIN*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Sine | Floating Point 5 |
| *SMLA* | Signed Multiply with Accumulate of 16 bit \* 16 bit values | ARMv5E |
| **SMLAL** | Signed Long (sign + 63 bit) Multiply with Accumulate | StrongARM |
| *SMLAL* | Signed Multiply with Accumulate of 16 bit \* 16 bit values, result is sign extended to 32 bits, then added to a 64 bit value. | ARMv5E |
| *SMLAW* | Signed Multiply with Accumulate of 32 bit \* 16 bit values | ARMv5E |
| *SMUL* | Signed Multiply of 16 bit \* 16 bit values | ARMv5E |
| **SMULL** | Signed Long (sign + 63 bit) Multiply | StrongARM |
| *SMULW* | Signed Multiply of 32 bit \* 16 bit values | ARMv5E |
| [*SQT*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Square Root | Floating Point 2 3 |
| [**STC**](https://www.heyrick.co.uk/assembler/coprocmnd.html#stc) | Co-processor data transfer | - |
| *STC2* | STC, *non-conditional* so more co-processor commands possible | ARMv5 |
| [*STF*](https://www.heyrick.co.uk/assembler/fpops.html#stf) | Store floating point value | Floating Point 1 3 |
| [**STM**](https://www.heyrick.co.uk/assembler/str.html#stm) | Store multiple registers | - |
| *STMIA* | Thumb: Store multiple registers | Thumb |
| [**STR**](https://www.heyrick.co.uk/assembler/str.html#str) | Store a register (32 bit) | - |
| *STR* | Thumb: Store register (32 bit?) | Thumb |
| [**STRB**](https://www.heyrick.co.uk/assembler/str.html#str) | Store a byte (8 bit) from a register | - |
| *STRB* | Thumb: Store byte (8 bit) | Thumb |
| **STRH** | Store a halfword (16 bit) from a register | StrongARM |
| *STRH* | Thumb: Store halfword (16 bit) | Thumb |
| **STRSB** | Store a signed byte (sign + 7 bit) from a register | StrongARM |
| **STRSH** | Store a signed half-word (sign + 15 bit) from a register | StrongARM |
| [**SUB**](https://www.heyrick.co.uk/assembler/mov.html#sub) | Subtract | - |
| *SUB* | Thumb: Subtract | Thumb |
| [*SUF*](https://www.heyrick.co.uk/assembler/fpops.html#binop) | Subtract | Floating Point 1 3 |
| [**SWI**](https://www.heyrick.co.uk/assembler/swi.html#swi) | Cause a SoftWare Interrupt | - |
| *SWI* | Thumb: SoftWare Interrupt | Thumb |
| [**SWP**](https://www.heyrick.co.uk/assembler/mov.html#swp) | Swap register with memory | ARM 3 |
| [*TAN*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Tangent | Floating Point 5 |
| [**TEQ**](https://www.heyrick.co.uk/assembler/cmp.html#teq) | Test Equivalence (notional EOR) | - |
| [**TST**](https://www.heyrick.co.uk/assembler/cmp.html#tst) | Test bits (notional AND) | - |
| *TST* | Thumb: Test bits | Thumb |
| **UMLAL** | Unsigned Long (64 bit) Multiply with Accumulate | StrongARM |
| **UMULL** | Unsigned Long (64 bit) Multiply | StrongARM |
| [*URD*](https://www.heyrick.co.uk/assembler/fpops.html#unop) | Unnormalised round | Floating Point 1 |
| [WFC](https://www.heyrick.co.uk/assembler/fpops.html#wfc) | Write FP control register | Floating Point 1 4 |
| [*WFS*](https://www.heyrick.co.uk/assembler/fpops.html#wfs) | Write FP status register | Floating Point 1 3 |

1. Pada ARM7500FE itu, instruksi FP ini disediakan oleh hardware.
2. Pada ARM7500FE itu, instruksi FP ini disediakan oleh perangkat lunak.
3. Instruksi FP ini diperlukan oleh IEEE (754-1985).
4. Instruksi FP ini hanya disediakan oleh hardware floating sistem point.
5. Instruksi FP ini disediakan untuk kompatibilitas mundur, dan ditiru.

Instruksi dalam huruf tebal adalah petunjuk inti ARM.

Instruksi dalam huruf miring disediakan oleh Floating Point Accelerator / Emulator.

Instruksi dalam huruf miring Dark Grey adalah aneka instrctions ARMv5.

Instruksi dalam huruf miring gelap Slate Grey VFP atau Thumb petunjuk.

Segala sesuatu yang lain adalah potongan-potongan yang layak termasuk, pilihan shift dan mnemonik assembler umum ...

Instruksi co-processor tercantum. Namun prosesor ARM yang digunakan dalam mesin RISC OS tidak mendukung co-prosesor, dan hanya fungsi co-processor maya hadir dalam chip dapat diakses. Ini menyediakan fasilitas untuk menyiapkan ARM, cache, MMU, dll ...

Instruksi Thumb adalah bagian dari instruksi ARM normal, dirancang untuk bekerja dalam mode 16 bit. Hal ini memungkinkan kode lebih kurang memori, dan untuk subsistem memori yang lebih murah untuk digunakan. Ini adalah kompromi antara kekuatan 32 bit ARM, dan efektivitas biaya. Sebelum Anda menggaruk kepala Anda dan berpikir "Huh? Tapi DIMM murah!", Ingat hidup agak berbeda ketika kita berbicara tentang perangkat spesialis tertanam, mulai dari pesawat ruang angkasa ke ponsel. Anyway, beberapa instruksi, seperti STR dan MOV yang digandakan untuk petunjuk Thumb sehingga tidak ada ambiguitas. The Vector Floating Point tampaknya menjadi superset dari komponen perangkat keras dari FPU asli. Hal-hal seperti POW dan COS tidak hadir, karena mereka dapat dilakukan dengan bantuan software - itu tidak akan sesuai dengan prinsip-prinsip RISC untuk mengabdikan silikon untuk hal-hal seperti itu.

DAFTAR PUSTAKA

* <https://oferiachacha.blogspot.co.id/2012/03/daftar-set-instruksi-8051-dan-contoh_22.html>
* <http://robisapoetra.blogspot.co.id/2013/11/bab-i-pendahuluan-a.html>
* https://www.heyrick.co.uk/assembler/qfinder.html