**Assembly coding ABC-3**

**1.. Mechanism behind Code Execution**

[a] Non-stop and Repetitive Instruction Cycles

|  |  |  |
| --- | --- | --- |
| **Instruction Cycle** ( life-span during which an instruction is executed) | | |
| logic level | architectural mechanism | phenomena in 51CPU (internal-access mode) |
| **PC** **code-space**  FETCH  DECODE  next I-cycle  EXECUTE | execution of the code-image residing in the code-space  1] FETCH \* retrieval of the 1st machine-code byte of current  instruction from code-space, according to PC;  setting;  \* PC++.  2] DECODE \* decoding of the 1st code-byte by which  -- operation of current inst. is identified;  -- further code bytes of current inst. to be  read is determined (inst. code-length, that is).  3] EXECUTE \* retrieval of successive code bytes of current instruction if any, according to PC setting;  \* PC setting updated accordingly, and thus  pointing to the inst. right after the current one;  \* instruction carried out, with effects of  execution reflected in the architecture as  aforementioned. | 1] FETCH  \* read the 1st code-byte of current inst.  from code-space according to PC;  \* PC++.  2] DECODE  \* identify for current instruction  -- the operation to be undertaken  -- the inst. code-length.  3] EXECUTE  \* read the 2nd code-byte if necessary and  PC adjusted accordingly;  \* read the 3rd code-byte if necessary and  PC adjusted accordingly;  \* execute |

[b] Code Execution Flow

\* sequential execution when not “detoured” by certain execution-flow controlling instructions;

\* instructions that would disturb the sequential execution of code:

1) function/procedure/subroutine ***calls***,

2) ***return*** from function-/procedure-/subroutine-call,

3) ***unconditional-branching*** to designated instruction,

4) ***conditional-branching*** to designated instruction;

\* the effect of all execution-flow control instruction all concerns with PC-modification.

1) information inherent in the code-byte associated with address computation for the targeted instruction

1.1) absolute-address code-byte computation by the assembler

|  |  |  |  |
| --- | --- | --- | --- |
| source line | machine code bytes | | note |
| LJMP label | opcode | operand(s): addr16 | \* a 3-byte long instruction  \* addr16: 16-bit target address associated with ***label***  \* addr16 := mm\*28 + nn (how is it determined? ☺) |
| 02H | mmH nnH |
| AJMP label | opcode | operand: addr11 | \* a 2-byte long instruction  \* X is an odd-number (i.e., X1[4]:= 1)  \* addr11 := YY\*23 + X1[7:5] (how is it determined? ☺)  \* range of branching?! |
| X1H | YYH |
| LCALL label | opcode | operand(s): addr16 | \* a 3-byte long instruction  \* addr16: 16-bit target address associated with ***label***  \* addr16 := ii\*28 + jj (how is it determined? ☺) |
| 12H | iiH jjH |
| ACALL label | opcode | operand: addr11 | \* a 2-byte long instruction  \* Y is an even-number (i.e., Y1[4]:= 0)  \* addr11 := ZZ\*23 + Y1[7:5] (how is it determined? ☺)  \* range of branching?! |
| Y1H | ZZH |

1.2) relative-address code-byte computation by the assembler

[ ***relative-address*** refers to the distance from the targeted-instruction to the branching instruction currently being executed ]

|  |  |  |  |
| --- | --- | --- | --- |
| source line | machine code bytes | | note |
| SJMP label | opcode | operand: rel8 | \* a 2-byte long instruction  \* **rel8**  mmH:= target address associated with ***label*** – PC\*  \* PC\*: current PC setting when SIMP is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| 80H | mmH |
| DJNZ Rn, label  JC label  JNC label  JZ label  JNZ label | opcode | operand: rel8 | \* a 2-byte long instruction  \* **rel8**  mmH:= target address associated with ***label*** – PC\*  \* PC\*: current PC setting when SJMP is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| XXH | mmH |
| JB bit8, label  JNB bit8,  label  JBC bit8,  label  DJNZ Rn, label DJNZ direct8, label CJNZ A, direct8, label CJNZ A, #data8, label  CJNZ Rn, #data8, label  CJNZ @Ri, #data8, label | opcode | opnd2 rel8 | \* a 3-byte long instruction  \* **rel8**  mmH:= target address associated with ***label*** – PC\*  \* PC\*: current PC setting when rel-JMP instruction is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| XXH | iiH jjH |
|  |  |

2) run-time computation of target-address for PC

2.1) run-time computation of target-address for PC with absolute-address in code bytes

|  |  |  |  |
| --- | --- | --- | --- |
| source line | machine code bytes | | effects |
| LJMP label | opcode | operand(s): addr16 | \* PC := mm\*28 + nn |
| 02H | mmH nnH |
| AJMP label | opcode | operand: addr11 | \* PC := YY\*23 + X1[7:5] |
| X1H | YYH |
| LCALL label | opcode | operand(s): addr16 | \* SP++; (SP) 🡨 PC\*L  \* SP++; (SP) 🡨 PC\*H  \* PC:= ii\*28 + jj |
| 12H | iiH jjH |
| ACALL label | opcode | operand: addr11 | \* SP++; (SP) 🡨 PC\*L  \* SP++; (SP) 🡨 PC\*H  \* PC := ZZ\*23 + Y1[7:5] |
| Y1H | ZZH |

2.2) run-time computation of target-address for PC by relative address in code bytes

|  |  |  |  |
| --- | --- | --- | --- |
| source line | machine code bytes | | note |
| SJMP label | opcode | operand: rel8 | \* a 2-byte long instruction  \* PC:= PC\* + rel8 = PC\* + mmH  \* PC\*: current PC setting when SIMP is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| 80H | mmH |
| DJNZ Rn, label  JC label  JNC label  JZ label  JNZ label | opcode | operand: rel8 | \* a 2-byte long instruction  \* PC:= PC\* + rel8 = PC\* + mmH if ***cond*** is TRUE  PC:= PC\* [i.e., unchanged] otherwise  \* PC\*: current PC setting when SJMP is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| XXH | mmH |
| JB bit8, label  JNB bit8,  label  JBC bit8,  label  DJNZ Rn, label DJNZ direct8, label CJNZ A, direct8, label CJNZ A, #data8, label  CJNZ Rn, #data8, label  CJNZ @Ri, #data8, label | opcode | opnd2 rel8 | \* a 3-byte long instruction  \* PC:= PC\* + rel8 = PC\* + mmH if ***cond*** is TRUE  PC:= PC\* [i.e., unchanged] otherwise  \* PC\*: current PC setting when rel-JMP instruction is being executed (, and so PC\*=?! ☺)  \* range of activity?! |
| XXH | iiH jjH |

**2.. Control of Code Execution Flow: An example**

Consider the 51 assembly program listed below.

|  |  |  |  |
| --- | --- | --- | --- |
| source lines | | | |
| org 0000h  mov SP, #60H ;75h-81h-60h  start:  mov R3, #3 ;7Bh-03h  mov R0, #50H ;78h-50h  mov R1, #40H ;79h-40h | next: ; :: 0009H  call arith ;71h-01h  mov @R0, A ;F6h  inc R0 ;08h  inc R1 ;09h  inc R1 ;09h  djnz R3, next ;DBh-F8h  mov @R0, #3 ;76h-03h  jmp start ;02h-03h-00h | arith: ; :: 0016H  push 01H ;C0h-01h  mov A, @R1 ;E7h  inc R1 ;09h  call add1 ;D1h-00h  inc R1 ;09h  call subb1 ;B1h-00h  pop 01H ;D0h-01h  ret ;22h | add1: ; :: 0022H  add A, @R1 ;27h  ret ;22h  sub1: ; :: 0024H  subb A, @R1 ;97h  ret ;22h  end |

Try filling out the entire table concerning the phenomena observed during the proceeding of instruction cycles

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inst.-  cycle | PC | 1st  fetch | PC | 2nd  fetch | PC | 3rd  fetch | PC | effects | SP | STACK status |
| 1 | 0000H |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |
| 21 |  |  |  |  |  |  |  |  |  |  |
| 22djnz |  |  |  |  |  |  |  |  |  |  |
| 23 |  |  |  |  |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 26 |  |  |  |  |  |  |  |  |  |  |
| 27 |  |  |  |  |  |  |  |  |  |  |
| 28 |  |  |  |  |  |  |  |  |  |  |
| 29 |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |
| 31 |  |  |  |  |  |  |  |  |  |  |
| 32 |  |  |  |  |  |  |  |  |  |  |
| 33 |  |  |  |  |  |  |  |  |  |  |
| 34 |  |  |  |  |  |  |  |  |  |  |
| 35 |  |  |  |  |  |  |  |  |  |  |
| 36 |  |  |  |  |  |  |  |  |  |  |
| 37 |  |  |  |  |  |  |  |  |  |  |
| 38 |  |  |  |  |  |  |  |  |  |  |
| 39 |  |  |  |  |  |  |  |  |  |  |
| 40djnz |  |  |  |  |  |  |  |  |  |  |
| 41 |  |  |  |  |  |  |  |  |  |  |
| 42 |  |  |  |  |  |  |  |  |  |  |
| 43 |  |  |  |  |  |  |  |  |  |  |
| 44 |  |  |  |  |  |  |  |  |  |  |
| 45 |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 46 |  |  |  |  |  |  |  |  |  |  |
| 47 |  |  |  |  |  |  |  |  |  |  |
| 48 |  |  |  |  |  |  |  |  |  |  |
| 49 |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |
| 51 |  |  |  |  |  |  |  |  |  |  |
| 52 |  |  |  |  |  |  |  |  |  |  |
| 53 |  |  |  |  |  |  |  |  |  |  |
| 54 |  |  |  |  |  |  |  |  |  |  |
| 55 |  |  |  |  |  |  |  |  |  |  |
| 56 |  |  |  |  |  |  |  |  |  |  |
| 57 |  |  |  |  |  |  |  |  |  |  |
| 58djnz |  |  |  |  |  |  |  |  |  |  |
| 59 |  |  |  |  |  |  |  |  |  |  |
| 60jmp |  |  |  |  |  |  |  |  |  |  |
| 61 |  |  |  |  |  |  |  |  |  |  |
| 62 |  |  |  |  |  |  |  |  |  |  |
| 63 |  |  |  |  |  |  |  |  |  |  |
| 64 |  |  |  |  |  |  |  |  |  |  |
| 65 |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 66 |  |  |  |  |  |  |  |  |  |  |
| 67 |  |  |  |  |  |  |  |  |  |  |
| 68 |  |  |  |  |  |  |  |  |  |  |
| 69 |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |
| 71 |  |  |  |  |  |  |  |  |  |  |
| 72 |  |  |  |  |  |  |  |  |  |  |
| 73 |  |  |  |  |  |  |  |  |  |  |
| 74 |  |  |  |  |  |  |  |  |  |  |
| 75 |  |  |  |  |  |  |  |  |  |  |
| 76 |  |  |  |  |  |  |  |  |  |  |
| 77 |  |  |  |  |  |  |  |  |  |  |
| 78 |  |  |  |  |  |  |  |  |  |  |
| 79 |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |
| 81 |  |  |  |  |  |  |  |  |  |  |
| 82 |  |  |  |  |  |  |  |  |  |  |
| 83 |  |  |  |  |  |  |  |  |  |  |
| 84 |  |  |  |  |  |  |  |  |  |  |
| 85 |  |  |  |  |  |  |  |  |  |  |

3..

|  |  |
| --- | --- |
| . . .  //adding data byte in block1 //with corresponding data //byte in block2, saving the  //sum in block3  . . .  ADD 3000H, 2000H, 1000H  ADD 3001H, 2001H, 1001H  ADD 3002H, 2002H, 1002H  ADD 3003H, 2003H, 1003H  ADD 3004H, 2004H, 1004H  ADD 3005H, 2005H, 1005H  ADD 3006H, 2006H, 1006H  ADD 3007H, 2007H, 1007H  . . .  //again, what if | Given a pseudo-CPU “TWN#1”  \* with 64KB memory data space (16 bits of address, that is): 0000H ~ FFFFH  \* 16 8-bit registers: R0~R15 |
| computing capacity of TWN#1  \* data transfer in byte – m2m, r2r, r2m, m2r  MOV destination, source  \* arithmetic-ops on byte – m&m, r&r, m&r, r&m  ADD sum, addend, adder  SUB difference, minuend, subtrahend  MUL product, multiplicand, multiplier  DIV quotient, remainder, dividend, divisor  INC opnd1  DEC opnd1  \* logic-ops on byte – m&m, r&r, m&r, r&m  AND opnd1, opnd2, opnd3  OR opnd1, opnd2, opnd3  XOR opnd1, opnd2, opnd3  CLR opnd1  NOT opnd1  RR opnd1  RL opnd2  \* no support of memory-indexing access  (i.e., memory units can not be accessed via index/pointer, but by address only)  \* no support of “initial value setting”  (i.e. mov R0, “999” not available  For initializing R0 with 999) |