32-Bit RISC Processor Pipeline

Instruction Encoding Scheme

- K.Kishorereddy

Encoding Scheme:

B31 B30 B29 B28 B27 B26	B25 B24 B23 B22 B21	B20 B19 B18 B17 B16	B15 B14 B13 B12 B11	B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
<---- Op Code --->	Destination Reg	Source Reg1	Source Reg 2	
			< ---------------- Immediate Value	--------------- >

Note:

- Please refer to the Example section for more specifics about encoding of each instruction.

Assembler Tables:

Instructions:

Instruction	Op-Code
MOV	XX0000
MVI	XX0001
ADD	XX0010
ADI	XX0011
SUB	XX0100
SUI	XX0101
AND	XX0110
ANI	XX0111
OR	XX1000
ORI	XX1001
LOAD	XX1010
STORE	XX1011
HLT	XX11XX

Note:

- Op-Codes of any of the Instructions require only 4-bit at maximum but to maintain the instruction size as 32-bits the Op-Code was made to be 6 bits.
- And the first two bits can be anything (hence the XX s in the above table) because only the last 4 bits of the Op-Code are considered for identifying the instruction.
- To use all the 16 numbers for instruction HLT was assigned 4 Op-codes (XX1100, XX1101, XX1110, XX111). Using anyone of those implies HLT instruction.

Registers:

Total of 32 registers are provided. And are addressed as R0, R1, …, R31.

Register	Encoding	Register	Encoding
R0	00000	R16	10000
R1	00001	R17	10001
R2	00010	R18	10010
R3	00011	R19	10011
R4	00100	R20	10100
R5	00101	R21	10101
R6	00110	R22	10110
R7	00111	R23	10111
R8	01000	R24	11000
R9	01001	R25	11001
R10	01010	R26	11010
R11	01011	R27	11011
R12	01100	R28	11100
R13	01101	R29	11101
R14	01110	R30	11110
R15	01111	R31	11111

Example Instructions:

The X s in the Binary format of the instruction imply that those bits does not affect the instruction. For simplicity they're taken as zeroes for generating hex format of the instruction.

- MOV R1, R2
  - R1 <- [R2]
  - XX 0000 00001 00010 XXXXXXXXXXXXXXXX
  - 00220000h
- MVI R1, 1111h
  - R1 <- 00001111h
  - XX 0001 00001 XXXXX 0001000100010001
  - 04201111h
- ADD R1, R2, R3
  - R1 <- [R2] + [R3]
  - XX 0010 00001 00010 00011XXXXXXXXXXX
  - 08221800h
- ADI R1, R2, 1111h
  - R1 <- [R2] + 00001111h
  - XX 0011 00001 00010 0001000100010001
  - 0c221111h

- SUB R1, R2, R3
    - R1 <- [R2] – [R3]
    - XX 0100 00001 00010 00011XXXXXXXXXXX
    - 10221800h
- SUI R1, R2, 1111h
    - R1 <- [R2] – 00001111h
    - XX 0101 00001 00010 0001000100010001
    - 14221111h
- AND R1, R2, R3
    - R1 <- [R2] AND [R3]
    - XX 0110 00001 00010 00011XXXXXXXXXXX
    - 18221800h
- ANI R1, R2, 1111h
    - R1 <- [R2] AND 00001111h
    - XX 0111 00001 00010 0001000100010001
    - 1c221111h
- OR R1, R2, R3
    - R1 <- [R2] OR [R3]
    - XX 1000 00001 00010 00011XXXXXXXXXXX
    - 20221800h
- ORI R1, R2, 1111h
    - R1 <- [R2] OR 00001111h
    - XX 1001 00001 00010 0001000100010001
    - 24221111h
- LOAD R1, 1111h(R2)
    - R1 <- [[R2]+00001111h]
    - XX 1010 00001 00010 0001000100010001
    - 28221111h
- STORE R1, 1111h(R2)
    - [R2]+00001111h <- [R1]
    - XX 1011 00001 00010 0001000100010001
    - 2c221111h
- HLT
    - XX 11XX XXXXX XXXXX XXXXXXXXXXXXXXXX
    - 30000000h

Machine code for the two sample programs in Hexadecimal:

- Program1 code to be loaded in memory
  - 04200000h
  - 28410000h
  - 00620000h
  - 1c420001h
  - 2c410001h
  - 00430000h
  - 1c420002h
  - 2c410002h
  - 00430000h
  - 1c420004h
  - 2c410003h
  - 00430000h
  - 1c420008h
  - 2c410004h
  - 00430000h
  - 1c420010h
  - 2c410005h
  - 00430000h
  - 1c420020h
  - 2c410006h
  - 00430000h
  - 1c420040h
  - 2c410007h
  - 00430000h
  - 1c420080h
  - 2c410008h
  - 30000000h
- Program2 code to be loaded in memory
  - 04200000h
  - 28410000h
  - 28610002h
  - 08831000h
  - 14840001h
  - 2c810001h
  - 0c210003h
  - 28410000h
  - 28610002h
  - 08831000h
  - 14840001h
  - 2c810001h
  - 04200000h
  - 28410001h
  - 28610004h
  - 20831000h
  - 2c810006h
  - 30000000h