# Instruction Format:

Big endian memory
SIZE BIT = 0 means 16-bit instruction
SIZE BIT = 1 means 32-bit instruction

1-OPERAND-type (16-bit instruction)  + push + pop

| SIZE BIT +OP<br>15:10<br><br>31:26 | OPERAND(RS)<br>9:6<br><br>25:22 | RD<br>3:0<br><br>19:16 |
|---|---|---|

2-OPERAND-type
ALU:

| SIZE BIT +OP<br>31 : 26 | RS<br>25 : 22 | RT<br>21 : 18 | RD<br>3:0 |
|---|---|---|---|

IMMEDIATE

| SIZE BIT +OP<br>31 : 26 | RS<br>25 : 22 | RD<br>19:16 | OFFSET<br>15:0 |
|---|---|---|---|

3-BRANCH-type

| SIZE BIT +OP<br>15:10<br><br>31:26 | TARGET<br>3 : 0<br><br>25:22 |
|---|---|

4-STD

| SIZE BIT +OP<br>31 : 26 | RS<br>25 : 22 | RT<br>21 : 18 | OFFSET<br>15: 0 |
|---|---|---|---|

Control signals:

Hlt , instruction size ,
call flush , return /int /RTI flush (the same signal ) ,
Alu src , branch flag ,
output signal , Alu control (its size will be determined  by the one who will implement it ) ,
wB destination selector  , mem read ,
mem write , write back ,
reg write , call flag ,
return flag , int flag ,
RTI flag , push flag  ,
pop  flag , mem data write

| | hlt | instruction size | call flush | return /int /RTI flush | Alu src | branch flag | output signal | Alu control | wB destination selector | mem read | mem write | write back selector | reg file write | call flag | return flag | int flag | RTI flag | push flag | pop flag | mem data write | Fetch flush |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NOP | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| HLT | 1 | 0 | 0 | 0 | xx | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| SETC | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| NOT | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| INC | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| OUT | 0 | 0 | 0 | 0 | xx | 0 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| IN | 0 | 0 | 0 | 0 | 10 | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| MOV | 0 | 1 | 0 | 0 | xx | 0 | 0 | x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| ADD | 0 | 1 | 0 | 0 | 00 | 0 | 0 | x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| SUB | 0 | 1 | 0 | 0 | 00 | 0 | 0 | x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| AND | 0 | 1 | 0 | 0 | 00 | 0 | 0 | x | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| IADD | 0 | 1 | 0 | 0 | 01 | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0(x) | 0 |
| PUSH | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | x | 0 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| POP | 0 | 0 | 0 | 0 | xx | 0 | 0 | x | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

| | hlt | instruction size | call flush | return /int /RTI flush | Alu src | branch flag | output signal | Alu control | wB destination selector | mem read | mem write | write back selector | reg file write | call flag | return flag | int flag | RTI flag | push flag | pop flag | mem data write | Fetch flush |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LDM | 0 | 1 | 0 | 0 | 01 | 0 | 0 | x | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LDD | 0 | 1 | 0 | 0 | 01 | 0 | 0 | x | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| STD | 0 | 1 | 0 | 0 | 01 | 0 | 0 | x | x | 0 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| JZ | 0 | 0 | 0 | 0 | xx | 1 | 0 | x | x | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JN | 0 | 0 | 0 | 0 | xx | 1 | 0 | x | x | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JC | 0 | 0 | 0 | 0 | xx | 1 | 0 | x | x | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JMP | 0 | 0 | 0 | 0 | xx | 1 | 0 | x | x | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CALL | 0 | 0 | 1 | 0 | xx | 0 | 0 | x | x | 0 | 1 | x | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| RET | 0 | 0 | 0 | 1 | xx | 0 | 0 | x | x | 1 | 0 | x | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| INT | 0 | 0 | 0 | 1 | 11 | 0 | 0 | x | x | 1 | 1 | x | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| RTI | 0 | 0 | 0 | 1 | xx | 0 | 0 | x | x | 1 | 0 | x | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

| SIZE BIT +OP | Instruction Size | OP CODE | Instruction | Control signals |
|---|---|---|---|---|
| 000000 | 16 | 0 | NOP | |
| 000001 | 16 | 1 | HLT | |
| 000010 | 16 | 2 | SETC | |
| 000011 | 16 | 3 | NOT | |
| 000100 | 16 | 4 | INC | |
| 000101 | 16 | 5 | OUT | |
| 000110 | 16 | 6 | IN | |
| 100111 | 32 | 7 | MOV | |
| 101000 | 32 | 8 | ADD | |
| 101001 | 32 | 9 | SUB | |
| 101010 | 32 | 10 | AND | |
| 101011 | 32 | 11 | IADD | |
| 001100 | 16 | 12 | PUSH | |
| 001101 | 16 | 13 | POP | |
| 101110 | 32 | 14 | LDM | |
| 101111 | 32 | 15 | LDD | |
| 110000 | 32 | 16 | STD | |
| 010001 | 16 | 17 | JZ | |
| 010010 | 16 | 18 | JN | |
| 010011 | 16 | 19 | JC | |
| 010100 | 16 | 20 | JMP | |
| 010101 | 16 | 21 | CALL | |
| 010110 | 16 | 22 | RET | |
| 010111 | 16 | 23 | INT | |
| 011000 | 16 | 24 | RTI | |

Questions:

1-

INT index

X[SP] ← PC; sp-=2;Flags reserved; (dedicated buffer for flags - execute o n 2-cycles )

PC ← M[index + 6] & M[index + 7]

RTI

sp+=2; PC ← X[SP]; Flags restored

2- PC ←R[ Rdst ]

The registers are 16-bits only while my PC is 32-bits ?

3- INT read and write in the same cycle like the regfile??

RST:

-signal to buffers, control unit , memory
-flush all stages
-tell the memory to return M[0],M[1]
-send a signal back to make the PC take M[0],M[1]
-Rest signal goes to the controller to make the hlt signal 0;

HLT:
-is known at decode stage in the control unit
-insert bubble after me and flush the instruction that was fetched after me
-freeze the pc (doesn't change as in the document)

LDD , STD:
-what is added to the offset is Rs as we have done
STD,PUSH:
-for them added a new mux before the alu to adjust the write data

RET:
-HLT signal is true to flush the fetch buffer
-plus signal (return/RTI/INT-flush) from the control which determines in (execute-memory) stages if it's a RET function and we should flush the fetch.
-note return-flush signal connected to the fetch/decode buffer is the or of return-flush signals in execute and memory (changed to fetch flush)

Call:
-HLT(Fetch flush ) signal is true to flush the fetch buffer
-plus signal (call-flush) from the control which determines in (execute) stages if it's a Call function and we should flush the fetch.

INT:
-HLT signal is true to flush the fetch buffer
-plus signal (return/RTI/INT-flush) from the control which determines in (execute-memory) stages if it's a INT function and we should flush the fetch.
-We will use the upper address which is (index+7) to make the operation of the INT and return the same
The ALU will add 7 to the index …
There will be a signal to the memory to read 32 bits instead of 16 bits…
The alu will store the flags in the upper bound in the flag reg when an int signal comes to it
The INT will be one operand the index will be from 16:19 and will enter to the mux as an offset
We will write in the First half cycle , and read in the second half
Push firstly then read

RTI:
-HLT signal is true to flush the fetch buffer
-plus signal (return/RTI/INT-flush) from the control which determines in (execute-memory) stages if it's a RTI function and we should flush the fetch.

Memory
Memory write and read are enables for the memory and we decided the size of read or write 32 or 16 with the call flag or the int flag

**Return, RTI and INt flushes the buffer till the memory stage……..**
**Fetch flush from Control unit Will be one  in call OR return OR INT OR RTI ..**

---