170070038
JAYESH
CHOUDHARY

EE 309 ASSIGNMENT-2

SINGLE INSTRUCTION PROCESSOR

AIM: To design a processor in which a single instruction can compute all the basic operations:
- Arithmetic and Logical instructions
- Data Transfer Instructions
- Branching Instructions

INSTRUCTION : Substract and branch if result is negative or zero

SUBLEQ A, B, C ⇒ M[B] = M[B] - M[A] and jump to C if M[B] now is negative or zero.

A and B are memory addsers
Let A & B be 16 bits C also 16 bits
Then data memory have 2^{16} locations
out of which some locations are resorved
Let us call them Z (M[Z] = 0)
and IP for storing address of next instruction. (IP+1)
T1, T2 for temporary purposes
$TA_1, TA_2, TA_3 - - - TA_{16}$ (only 1 bit is 1 in whole seq)
(0000000000000001, 0000000000000010 and so on)
TB stores FFFF

since the instruction is of 48 bits and I-memory also of 48 bits ∴ 1 word instruction
and hence IP = IP+1 for next instruction

1) SUBTRACTION: M[B] = M[B] - M[A]
Instruction : SUBLEQ A, B, IP

2) ADDITION: M[B] = M[B] + M[A]
Instruction : SUBLEQ A, Z, IP (M[Z] = M[Z] - M[A] = -M[A])
SUBLEQ Z, B, IP (M[B] = M[B] - M[Z] = M[B] + M[A])
SUBLEQ Z, Z, IP (M[Z] = 0 To again put zero)

3) MOVING: M[B] = M[A]
Instruction : SUBLEQ A, Z, IP (M[Z] = -M[A])
SUBLEQ B, B, IP (M[B] = 0)
SUBLEQ Z, B, IP (M[B] = -(-M[A]) = M[A])
SUBLEQ Z, Z, IP (M[Z] = 0)

4) JUMP (UNCONDITIONAL):
Instruction : SUBLEQ Z, Z, J M[Z] = 0
Since result is = 0 ∴ jump location is specified by J.

5) BRANCH EQUALITY JUMP : JUMP TO J if M[A] = M[B]
Here we need to use temporary values T1 and T2 be temporary location
Instruction :

SUBLEQ A, Z, IP M[Z] = -M[A]
SUBLEQ T1, T1, IP M[T1] = 0
SUBLEQ Z, T1, IP M[T1] = M[A]
SUBLEQ Z, Z, IP M[Z] = 0
SUBLEQ B, Z, IP M[Z] = -M[B]
SUBLEQ T2, T2, IP M[T2] = 0
SUBLEQ Z, T2, IP M[T2] = M[B]
SUBLEQ Z, Z, IP M[Z] = 0.

```
       SUBLEQ  T1, T2, J1          M[T2] = M[T2] - M[T1]
       SUBLEQ  Z, Z, J2            M[Z] = 0  (if no jump)
J1:    SUBLEQ  B, T1, J            M[T1] = M[T1] - M[B]
J2:    - - - - -
       - - - - - .
J:     - - - - - .
```

First we move M[A] → M[T1] and M[B] → M[T2] so-that A and B could be used further.

If M[T2] - M[T1] is +ve, then no need for jump so we move to J2.

If M[T2] - M[T1] is -ve or zero then jump to J1 where we check if M[T1] - M[T2] is also (-ve or zero) or not. But T2 has been modified. So we use M[B]. If both conditions satisfied then we move to J.

[Crux: If $a-b \leq 0$ and $b-a \leq 0$ then $b-a=0$ or $\underline{a=b}$.]

6) CLEARING : M[B] = 0.

Instruction : SUBLEQ B, B, IP   M[B] = 0.

7) COPYING : M[T1] = M[A]

Instruction :

```
SUBLEQ  A, Z, IP       M[Z] = - M[A]
SUBLEQ  T1, T1, IP     M[T1] = 0
SUBLEQ  Z, T1, IP      M[T1] = M[A]
SUBLEQ  Z, Z, IP       M[Z] = 0.
```

Data is copied to temporary location T1.

8) MOVING IMMEDIATE (16 bits):

- Let the sequence be (10010000 0000 0000)

In data memory there will be reserved location to store following values:

0000000000000001 (TA_1)
0000000000000010 (TA_2)
0000000000000100 (TA_3)

and so on till 1000000000000000 (TA_{16})

so we have reserved 16 locations for TA_1, TA_2, TA_3 till TA_{16}.

We need to move it to B.

INSTRUCTION:	SUBLEQ B, B, IP	M[B] = 0
	SUBLEQ TA_{16}, Z, IP	M[Z] = $-$M[TA_{16}]
	SUBLEQ Z, B, IP	M[B] = M[TA_{16}]
	SUBLEQ Z, Z, IP	M[Z] = 0
	SUBLEQ TA_{13}, Z, IP	M[Z] = $-$M[TA_{13}]
	SUBLEQ Z, B, IP	M[B] = M[TA_{16}]+M[TA_{13}]
	SUBLEQ Z, Z, IP	M[Z] = 0.

9) NOT : (BITWISE) (for B) (Lets say $\bar{B}$)

- Let a location in memory be reserved for $(FFFF)_{\text{hexadecimal}}$ lets say TB.

INSTRUCTION:	SUBLEQ TB, Z, IP	M[Z] = $-$M[TB]
	SUBLEQ T1, T1, IP	M[T1] = 0
	SUBLEQ Z, T1, IP	M[T1] = M[TB]
	SUBLEQ Z, Z, IP	M[Z] = 0
(T1 stores $\bar{B}$) - - - - - ->	SUBLEQ B, T1, IP	M[T1] = M[T1]$-$M[B]
	SUBLEQ T1, Z, IP	M[Z] = $-$M[T1]
	SUBLEQ B, B, IP	M[B] = 0
	SUBLEQ Z, B, IP	M[B] = $-$($-$M[T1]) = M[T1]
	SUBLEQ Z, Z, IP	M[Z] = 0.

10) LEFT SHIFT : for B $M[B]=M[B]\times 2$

```
INSTRUCTION :   SUBLEQ B,Z, IP        -M[B] = M[Z]
                SUBLEQ Z,B, IP        M[B]=M[B]-(-M[B])
                SUBLEQ Z,Z, IP        M[Z]=0
```

11) MULTIPLY :

Multiplication can be done using sepetitive addition and keeping track of another operand while subtracting it.

$$a \times b = \underbrace{a+a+a....}_{b\text{times}}$$

```
loop:  ADD  A,A                    SUB  B, TA1
       SUB  B, TA1        :-       (if +ve then continue)
       goto loop if B>1            ADD  A,A
                                   (go back)
```

To retain the identity of A and B, we can use temporary T1 and T2.

```
J1 :   SUBLEQ  TA1 , B, J         M[B] = M[B] - 1
       SUBLEQ  A, Z, IP           M[Z] = -M[A]
       SUBLEQ  Z, A, IP           M[A] = 2M[A]
       SUBLEQ  Z, Z, J1           M[Z]=0   jump back to J1
J :    _ _ _ _ _
       _ _ _ _ _
```

We can use T1 and T2 to retain identity of A and B.

12) AND (BITWISE)

For 1 bit AND = MULTIPLY

so we do MULTIPLY, then we right shift it then perform MULTIPLY and so on.

After 1st MULTIPLY, we use the look up table or predefined values in memory and find the value. and store it in Temporary location. Then using. Left shifting we can accumulate all the bits to get bitwise AND.

```
T1 = 0
Multiply
Based on last bit store in T1
Right Shift
Multiply
Based on last bit → Left shift → add in T1
Right Shift
  ⋮ ⋮  and so on.
```

13) RIGHT SHIFT

Right shift could be implemented by division by 2.

14) DIVISION

Division could be implemented by repetitive subtraction and maintaing a counter.

a ÷ b.

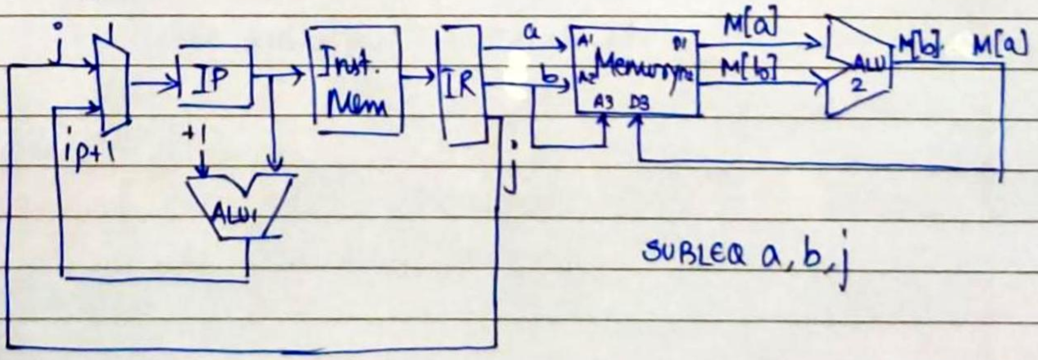
(keep on doing a=a-b till a is +ve)
(increment temporary location T2)

C = a ÷ b

```
J1:  SUBLEQ  B, A, J2       M[A] = M[A] - M[B]   Jump to J2 if -ve or zero
     SUBLEQ  TA1, Z, IP     M[Z] = -1
     SUBLEQ  Z, C, IP       M[C] = M[C] + 1
     SUBLEQ  Z, Z, J1       M[Z] = 0
J2 : BEQ  A, Z, J3          if M[A] = 0 then jump to J3.
                               (A now have A-B),
J  : - - - - -
     - - - - -              (rest of the code)
     - - - - -

J3 : [ Mov 1 in C           ( if A = B then we need
       Jump back to J ]       to set C = 1 )
```

Now we have AND, NOT making up NAND which is universal gate thus all Logical operations can be performed.
We can do data transfer and all branching Let it be Conditional or unconditional.
We also have all basic arithmetic operations
Thus, using this 1 instruction, we can do almost every computation.



SUBLEQ a, b, j

IR = Instruction_Memory Data.

IP → Inst_Mem_A, ALU1_A
+1 → ALU_B
ALU → mux_IN_0
Inst_Mem_D$_{15-0}$ → mux_in_1
Inst_Mem_D$_{31-16}$ → Data_Mem_A2
Inst_Mem_D$_{47-32}$ → Data_Mem_A1
Data_Mem_D1 → ALU2_A
Data_Mem_D2 → ALU2_B
ALU2 → Data_Mem_D3
Inst_Mem_D$_{31-16}$ → Data_Mem_A3
mux_out → IP

Here
Inst_Mem_D
= IR.

Note: We are reading several date from memory and writing back at the same time in 1 cycle

Brief Architecture : Inst_Memory = 48 bit
IR = 48 bit
IP = 48 bit (16 + 16 + 16)
no bit for operation.
Data Memory = 16 bit

- M_address1, M_address2 = Reading address
- M_address 3 = Writeback address
- M_data3 = Writeback data