2021-22 CSE-305

Section: B

Tabib

5. (a)

(i) Simplicity favors regularity. (!)

(ii) Smaller is faster.

(iii) Simplicity favors regularity.

(iv) Good design requires good compromise.

(v) Make the common case faster.

5(b)

(i) out of syllabus??

(ii) $\frac{100}{60} = \frac{98}{n} + 2$

$\Rightarrow n = -270$, so not possible. } out of syllabus??

(c) out of syllabus?

6. (a)

```
fib:  addi  $sp, $sp, -12
      sw    $ra, 8($sp)
      sw    $s0, 4($sp)
      sw    $a0, 0($sp)
      bgt   $a0, $zero, L1
      add   $v0, $zero, $zero
      j     ret

L1:   addi  $t0, $zero, 1
      bne   $t0, $a0, rec
      add   $v0, $zero, $t0
      j     ret

rec:  subi  $a0, $a0, 1
      jal   fib
      add   $s0, $v0, $zero
      subi  $a0, $a0, 1
      jal   fib
      add   $v0, $v0, $s0
```

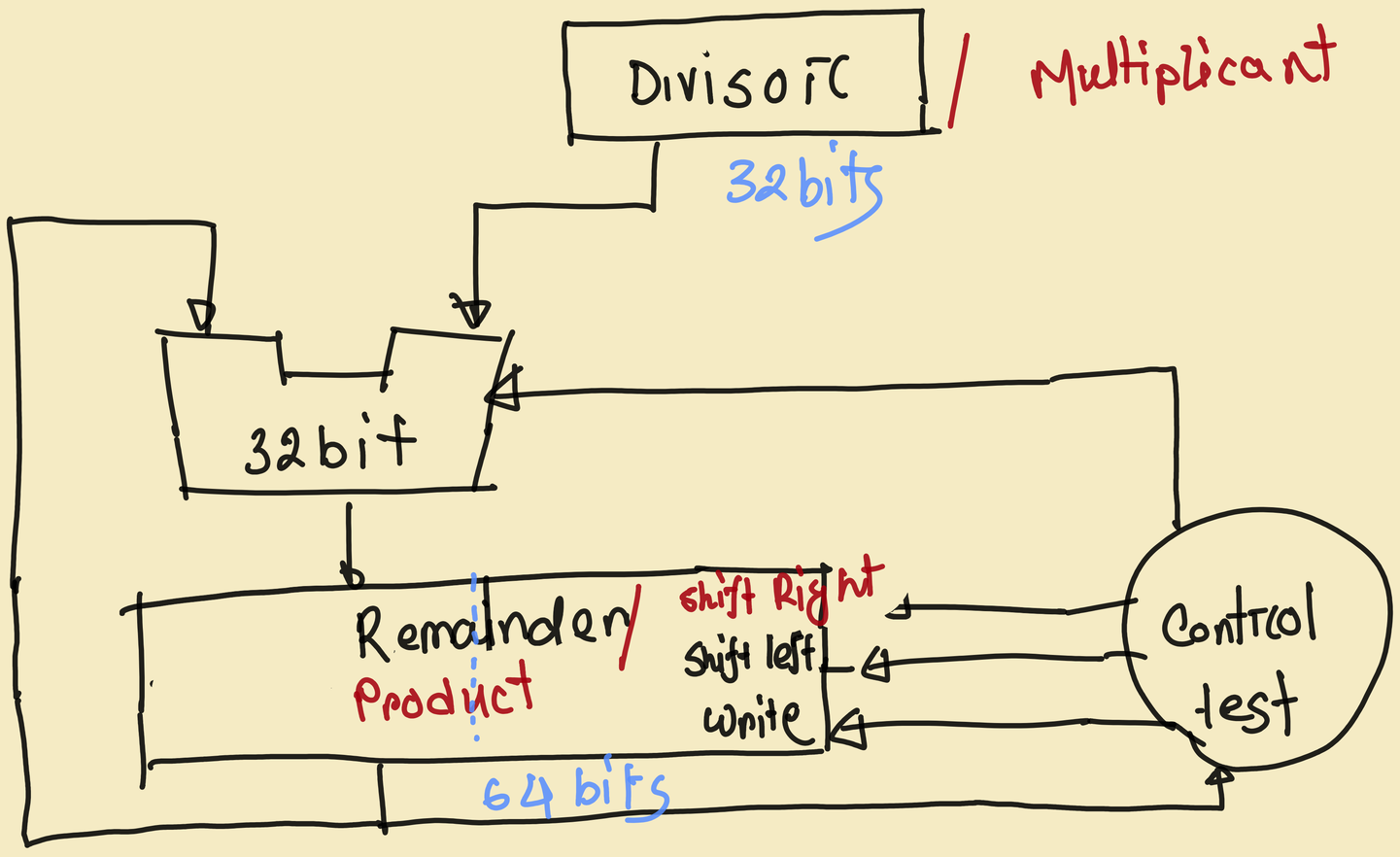
ret : lw $a0, 0($sp)
lw $s0, 4($sp)
lw $ra, 8($sp)
addi $sp, $sp 12
jr $ra

6. (b)

(i) first 4 bit of address would be same as PC address.
So, not possible.

(ii) not possible too. out of range covered by 16 bit.

5. (c)



for multiplication:

the product is initially at the left half of the product register (64 bits). Multiplier is at the second half. While multiplication process starts, after multiplying LSB of multiplier with the multiplicant, the multiplier is right shifted each time and the updated product's rightmost bit also shifted 1 bit right.

for division:

in remainder register, first 32 bits are initially 0 and the right-most 32 bits are populated with the dividend. It is left shifted each time. And compare the remainder with the divisor. If remainder_divisor >0, the this subtracted (left new) result is stored on that very left side and a 1 is placed as quotient at the rightmost bit of 64 bit remainder register's right half. Otherwise, restore the past remainder on the left and place 0 at the rightmost bit of right side of 64 bit rx.

7 → out of syllabus

8: (a)

(1)

P1:

total clock cycle required =

$$\sum_{\text{class } x}^{x: A,B,C} (CPI) * \text{instr.cnt}$$

$$= 3 \times (200{,}000 \times 25\%) + (3 \times 200{,}000 \times 60\%) + 2 \times (200{,}000 \times 15\%)$$

$= 57\,0000$

P2:

total clock cycles required:

$= 2\times(200{,}000\times 25\%) + 4\times(200{,}000\times 60\%) + 1\times(200{,}000\times 15\%)$

$= 610000$

P1:

$$\text{CPU time} = \frac{\text{tot cycle}}{\text{clock rate}}$$

$$= \frac{570000}{2.7\times 10^9}$$

$$= 2.1\times 10^{-4}\ s$$

P2:

$$\text{CPU time} = \frac{\text{tot cycle}}{\text{clock rate}}$$

$$= \frac{610000}{3.5\times 10^9} = 1.74\times 10^{-4}\ s$$

P_2 is faster.

(iii)

To run two times faster,

$$\text{CPU time for } P_2 = \frac{1.74 \times 10^{9}}{2} = \frac{2 \times 200{,}000 \times 25\% + 4 \times 200{,}000 \times 60\% + x \times 200{,}000 \times 15\%}{3.5 \times 10^{9}}$$

$$\Rightarrow x = -9.18$$

not possible

(b) actual exponent in floating point representation:

exp bias

$$= (2^8 - 1 - 1) - (2^7 - 1)$$

$$= +127$$

highest number to be represented $\approx 1.11\ldots. \approx 2$

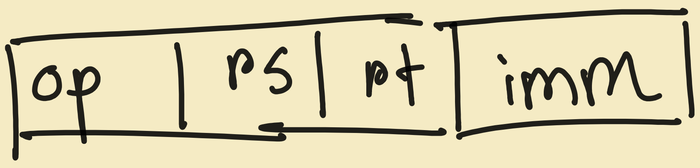
$$\therefore \text{highest number} \approx 2.0 \times 2^{+127}$$

$$\approx 3.4 \times 10^{38}$$

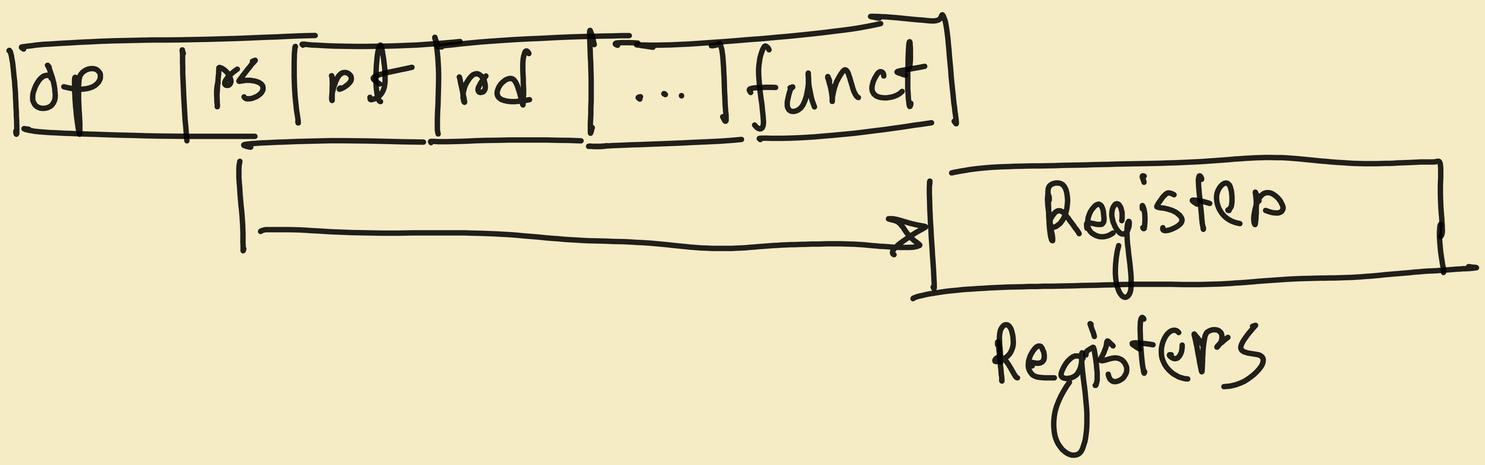
So. the n should be $\approx 3.4 \times 10^{38}$

8: © Addressing modes:

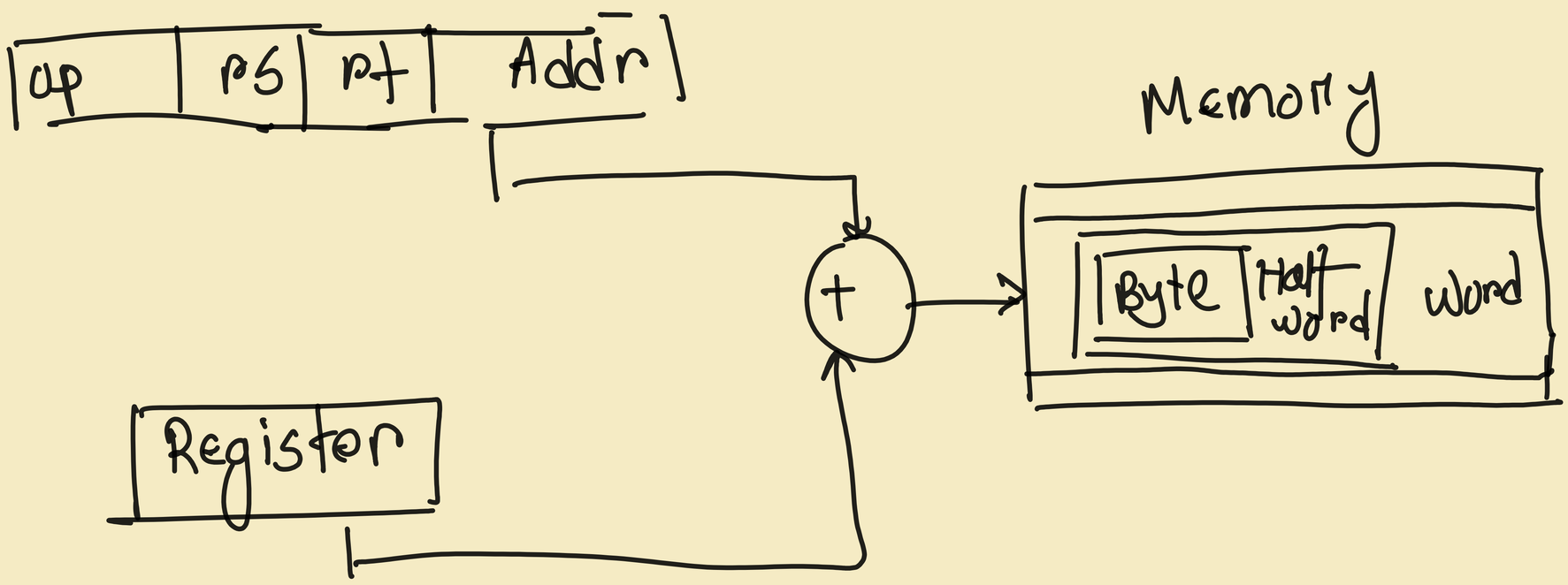
Immediate addressing: operand is a constant within the instruction itself.



Register addressing: operand is a register.

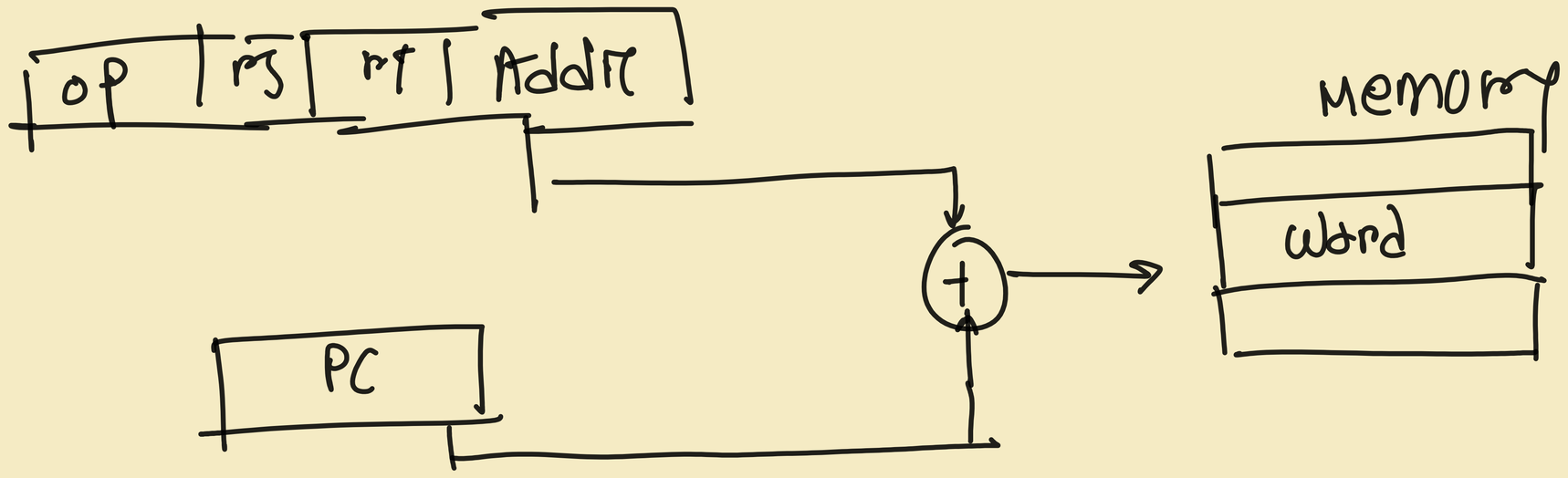


Base Addressing: operand is at the memory location whose address is the sum of a register and a constant in the instruction.

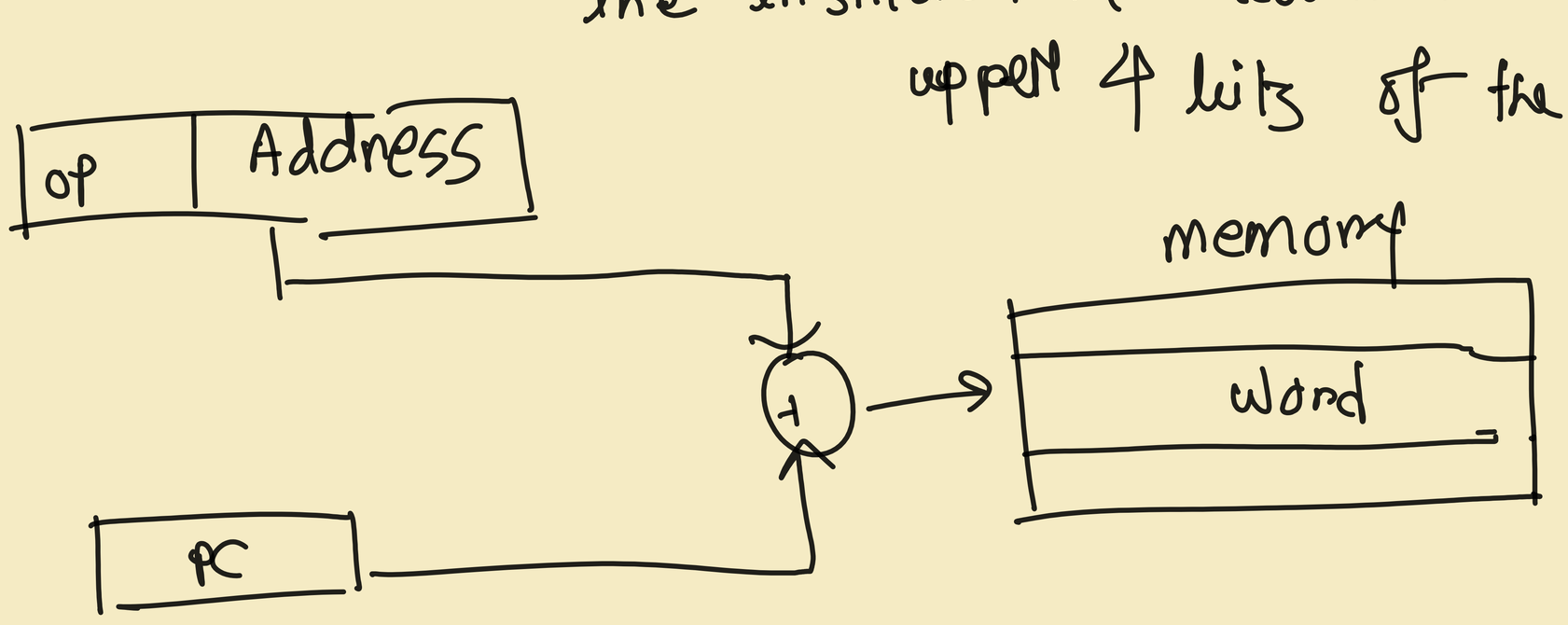


PC-Relative Addressing:

branch addr is the sum of PC and a constant in the instruction



5. Pseudo-direct addressing: where the jump addn is the 26 bits of the instruction concatenated with the upper 4 bits of the PC.



Section: B

2020-21

11. (a)

- ❑ Design for Moore's Law
- ❑ Use Abstraction to Simplify Design
- ❑ Make the Common Case Fast
- ❑ Performance via Parallelism
- ❑ Performance via Pipelining
- ❑ Performance via Prediction
- ❑ Hierarchy of Memories
- ❑ Dependability via Redundancy

} Brief description of each

10. (b)

S_3	S_2	S_1	S_0	Required Functions
0	0	0	0	$F = A$
0	0	0	1	$F = A - 1$
0	0	1	0	$F = A + B + 1$
0	0	1	1	$F = A + B$
0	1	0	0	$F = A - B$
0	1	0	1	$F = A - B - 1$
0	1	1	0	$F = A + 1$
0	1	1	1	$F = A$
1	0	0	X	$F = A'$
1	0	1	X	$F = AB'$
1	1	0	X	$F = A \odot B$ [XNOR]
1	1	1	X	$F = A \vee B$ [OR]

logical

X	Y	Z0
A	0	0
A	1	0
A	B	1
A	B	0
A	B'	1
A	B'	0
A	0	1
A	1	1
A'	0	0
AB'	0	0
A	B'	0
A	0	0

OR B

A

$X: S_3'A + S_3S_2'S_1'A' + S_3S_2'S_1AB' + S_3S_2S_1B$

S_3S_2 \ S_1S_0	00	01	11	10
00	1 (0)	1 (1)	1 (3)	1 (2)
01	1 (4)	1 (5)	1 (7)	1 (6)
11	(12)	(13)	(15)	(14)
10	(8)	(9)	(11)	(10)

$Y: S_3'S_0 + S_3'S_2'S_1B + S_2S_1'B'$

20

S_3S_2 \ S_1S_0	00	01	11	10
00	(0)	(1)	1 (3)	(2)
01	1 (4)	(5)	1 (7)	1 (6)
11	(12)	(13)	(15)	(14)
10	(8)	(9)	(11)	(10)

$Z_0: S_3'S_1S_0 + S_3'S_2S_1 + S_3'S_2S_0'$

$AL = S_3S_2$ → for logical

in every carry, $s_3 s_2$ will be AND. with the carry. It ensures carry flow only for arithmetic operations

11. (a)

$$\text{SPEC}_{\text{ratio}} = \frac{\text{reference}}{\text{measured}}$$

geometric mean:

$$\left(\frac{9770}{x} \times \frac{10490}{527} \times \frac{12100}{586} \times \frac{20720}{109} \times \frac{7020}{470} \times \frac{6900}{275}\right)^{\frac{1}{6}} = 28.74$$

$$\Rightarrow x = 507.63$$

11. (b) 3 micro operations are needed for addition in scratchpad memory employed processor.

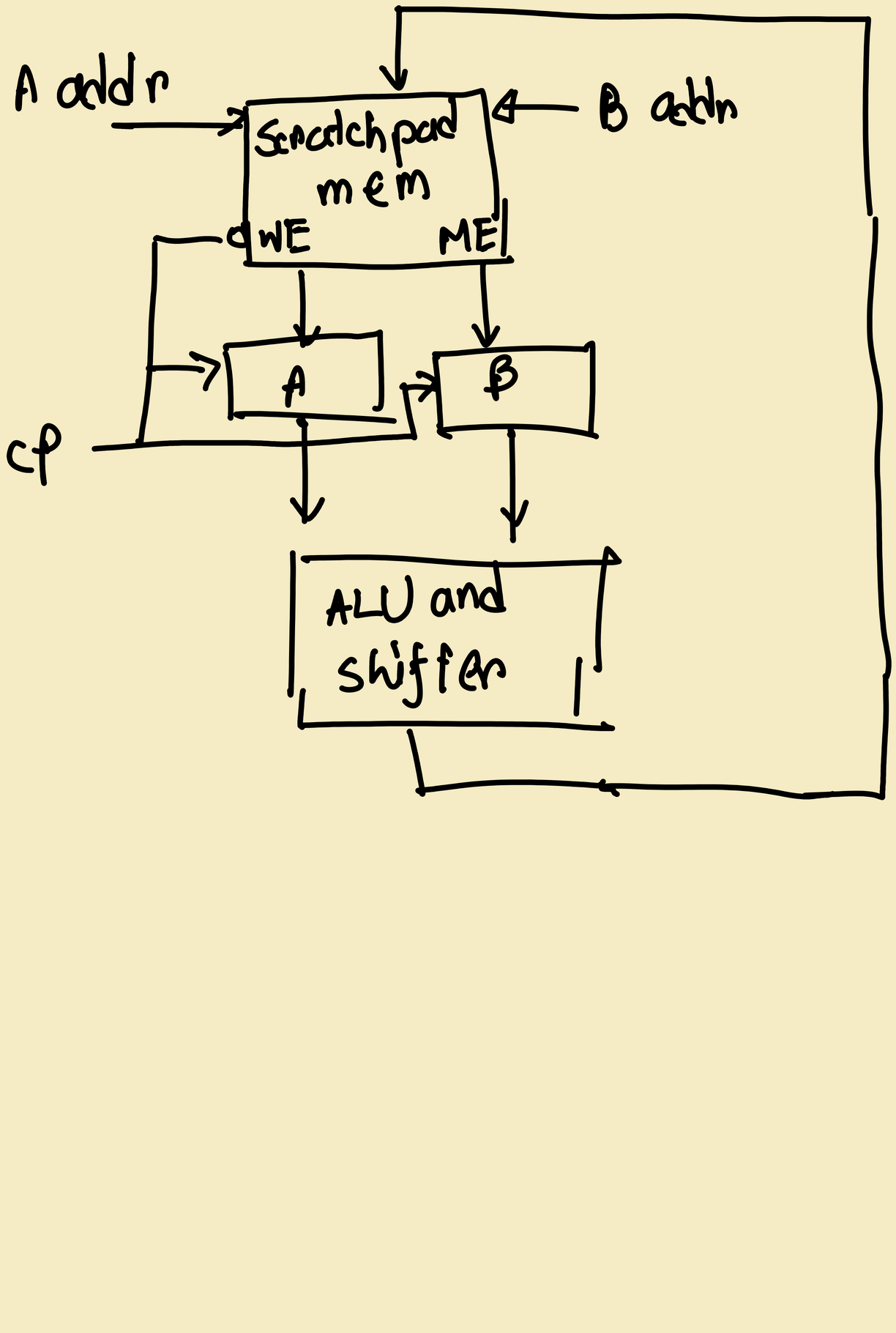
$$R_1 \leftarrow R_2 + R_3$$

Read : A ← M[010]

" B ← M[011]

write : M[001] ← A+B

Here two operations are needed to read values from R_2 and R_3. since address is coming from single path and then choosing destination register each time. we can reduce the operation placing two address bus and separate selector line for each registers.



(c)

```
testfunc:
        addi    $sp, $sp, -16
        sw      $ra, 12($sp)
        sw      $a2, 8($sp)
        sw      $a1, 4($sp)
        sw      $a0, 0($sp)

        addi    $s1, $zero, 0
        addi    $s2, $zero, 0

do-while: slt   $t0, $a1, $s1   // (i > b)
        beq     $t0, $zero, L1
        add     $s2, $s2, $a2
L1:     addi    $s1, $s1, 1
        slt     $t1, $s1, $a0
        bneq    $t1, $zero, dowhile
```

$s1 ← var i

$s2 ← var res

```
add  $v0, $s2, $zero

lw   $a0, 0($sp)
lw   $a1, 4($sp)
lw   $a2, 8($sp)
lw   $ra, 12($sp)
addi $sp, $sp, 16
jr   $ra
```

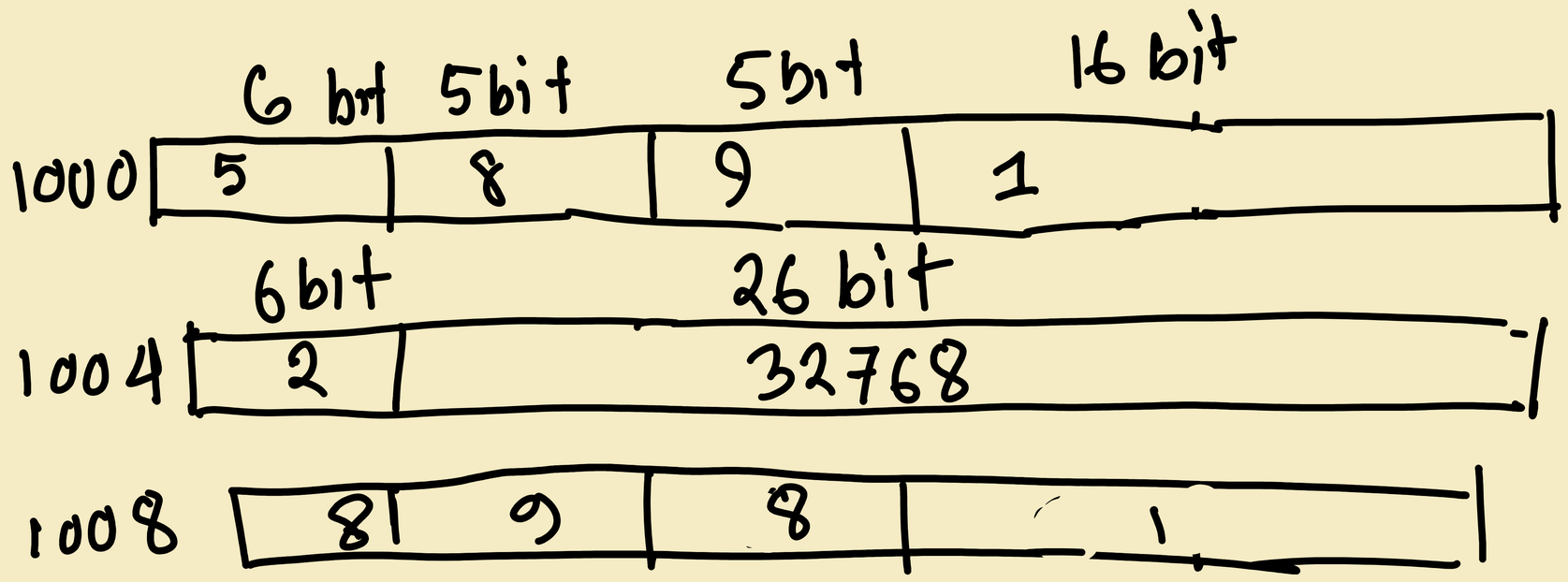
12. (a) three types of instruction format. It is a compromise between providing more larger addresses and constants in instructions and keeping all instructions the same length.

(b) Avg clock cycle for program A, B:

$$(\text{Avg CPI})_A: \quad \frac{2\times3 + 1\times1 + 4\times2 + 2\times4}{2+1+4+2} = 2.56$$

$$(\text{Avg CPI})_B = \frac{3\times3 + 2\times1 + 3\times2 + 2\times4}{3+2+3+2} = 2.5$$

11. (b)

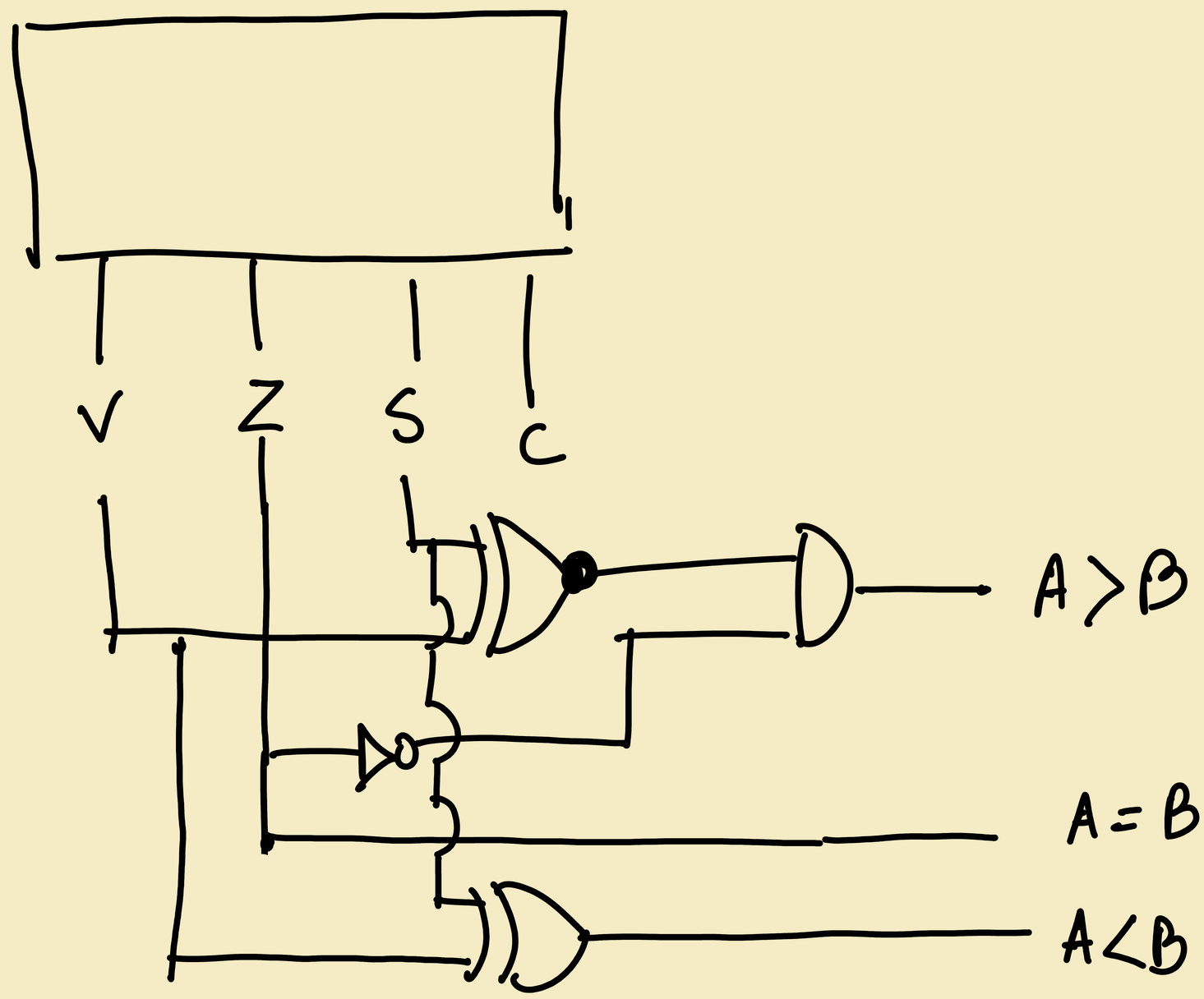


12. (c)

guard bit = 0 → always truncate

- Guard and round digits, and sticky bit
- When computing result, assume there are several extra digits available for shifting and computation. This improves accuracy of computation.
- Guard digit: first extra digit/bit to the right of mantissa -- used for rounding addition results
- Round digit: second extra digit/bit to the right of mantissa -- used for rounding multiplication results
- Sticky bit: third extra digit/bit to the right of mantissa – used for resolving ties such as 0.50...00 vs. 0.50...01

13. (d)



12. (a) IEEE std wanted to ensure simplified sorting while representing floating point numbers. neg. exponents can pose a problem to it. So, via biasing, exponents are kept positive.

(b) s → 1 exp → 7 frx → 24

$$\text{bias} : 2^6 - 1 = 63$$

(c) smallest : $1.00\underbrace{\ldots\ldots}_{24} \times 2^{1-63}$

$$= 1.00\ldots 0 \times 2^{-62}$$

largest : $1.\underbrace{11\ldots\ldots1}_{24} \times 2^{126-63}$

$$\approx 2.0 \times 2^{63}$$

(iv) smallest denormalized no : $0.\underbrace{00\ldots\ldots1}_{24\text{ bits}} \times 2^{-63}$

$$= 1.0 \times 2^{-63-24}$$

$$= 1.0 \times 2^{-87}$$

13. (b)

(PC+4) + sign extended offset<<2

13. (c)

```
add.d   $f16, $f0, $f2
sub.d   $f18, $f16, $f4
c.gt.d  $f18, $zero
bc1f    exit
mul.d   $f20, $f10, $f12
div.d   $f22, $f20, $f8
swc1    $f22, 0($f10)
exit:
```

$f0 ← a
$f2 ← b
$f4 ← c
$f6 ← d
$f8 ← w
$f10 ← x
$f12 ← y
$f14 ← z