高雄大學資訊工程系　計算機組織　期中考

學號：　　姓名：

1. (15%)MIPS has five addressing modes: immediate addressing, register addressing, base addressing, PC-relative addressing, and pseudo-direct addressing. Give their format and explain them. (各 3 分)

Register	Add R4,R3	R4 ← R4+R3
Immediate	Add R4,#3	R4 ← R4+3
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]

Pseudo-direct

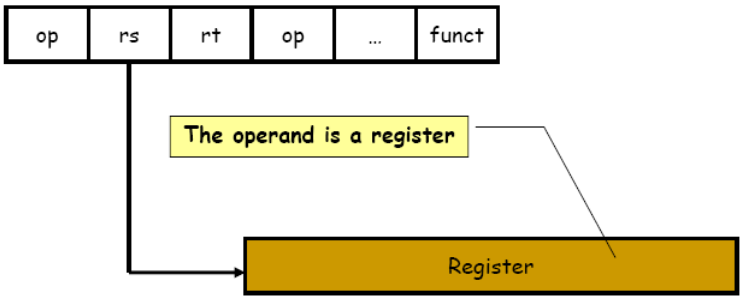
畫圖，或是文字說明

- Immediate addressing

Example: addi $2, $3, 4

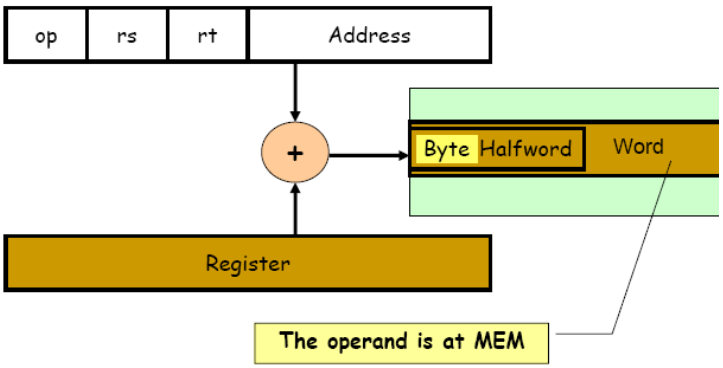
The operand is a constant within the instruction itself

- Register addressing



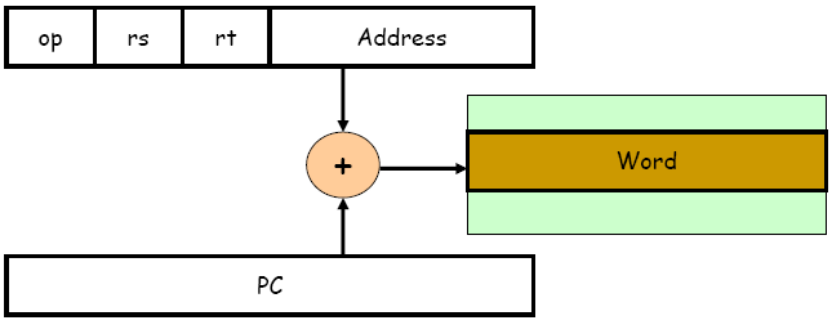
Example : add $r1, $r2, $r3

- Base addressing



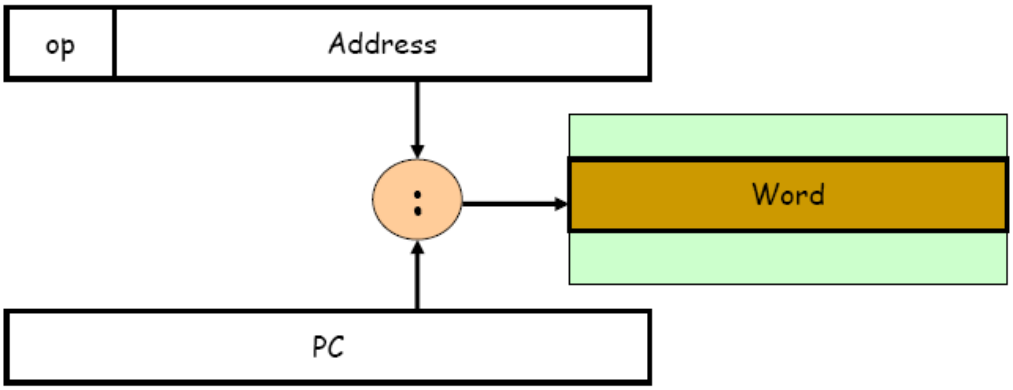
Example : lw $2, 100($3)

- PC-relative addressing



Example : beq $2, $3, 100

- Pseudodirect addressing



Example : j 100

2. (6%)Explain the Amdahl's Law by an example.

- Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected, then:

$$\mathbf{ExTime(E)} = ((1-F) + F/S) \times \text{ExTime(without E)}$$

$$\mathbf{Speedup(E)} = \frac{1}{(1-F) + F/S}$$

3. (8%)Explain four different styles of instruction sets: accumulator, memory-memory, stack, and load-store. You can give some example for explication. (各 2 分)

- Accumulator (1 register):

1 address: add A //acc ← acc + mem[A]
1+x address: addx A//acc ← acc + mem[A+x]

- Stack:

0 address: add //tos ← tos + next

- Memory-Memory:

????

- Load/Store: (a special case of GPR)

```
3 address:  add    $ra,$rb,$rc    //$ra ← $rb + $rc
            load   $ra,$rb        //$ra ← mem[$rb]
            store  $ra,$rb        //mem[$rb] ← $ra
```

4. (4%)Please write the internal data representation for the following declarations in a C program:
   A. int x
   B. float y

32-bit two's complement
IEEE 754 single precision floating point

5. (10%)Write -8_{10} in the following representations using 4 bits. If it is not possible to write it in the representation, write N/A. (各 2)

Signed Magnitude	N/A
1's Complement	N/A
2's Complement	1000
Excess 8	0000
Excess -3	N/A

6. (9%) Convert -17.5 to IEEE single precision. Convert 510_{10} into 32-bit two's complement binary number. Convert 10.4_{10} to binary. Label bits clearly. (各 3)

   1 1000 0011 0001 1000 0000 0000 0000 000

   0000 0000 00000 0000 0000 0001 1111 1110

   $10.4_{10} = 1010.\underline{0110}_2$

7. (4%) What do the following numbers mean according to IEEE 754 standard? If it is not a number, please write NaN. (各 2)

   A. 0 1111 1111 0000 0000 0000 0000 0000 000
   B. 0 1111 1111 0011 0000 0000 0000 0000 000

   Positive infinity,
   NaN

8. (4%) What is an instruction set architecture? (各 1)

   - Organization of programmable storage
     - registers
     - memory: flat, segmented
     - Modes of addressing and accessing data items and instructions
   - Data types and data structures
     - encoding and representation (next chapter)
   - Instruction formats
   - Instruction set (or operation code)
     - ALU, control transfer, exceptional handling

9. (6%) We want to add 0xABABCDCD to register $s1. However, in MIPS, the immediate field of an I-format instruction is only 16 bits. How can we complete such operation? Please write MIPS instructions.

```
lui     $at, 0xABAB
ori     $at, $at, 0xCDCD
add     $s1,$s1,$at
```

10. (4%) Please describe the detail behavior of instruction “jal”. (各 2)

(1) Store the the return address (PC +4) at $ra
(2) set PC = procedure_addre

11. (12%) Please describe the steps for execution of a procedure or subroutine.

(各 2)

- The program (caller) places parameters in places where the procedure (callee) can access them
- The program transfers control to the procedure
- The procedure gets storage needed to carry out the task
- The procedure carries out the task, generating values
- The procedure (callee) places values in places where the program (caller) can access them
- The procedure transfers control to the program (caller)

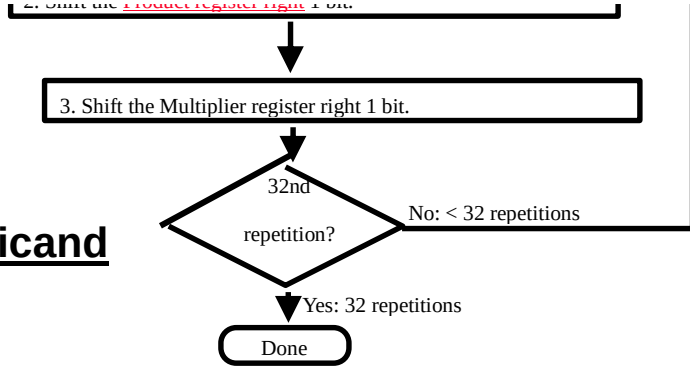
Or

- Caller puts parameter values in $a0-$a3
- Caller uses a jal X to jump to procedure X (callee)
- Callee performs calculations
- Callee place results in $v0-$v1
- Callee returns control to the caller using jr $ra

12. (6%) If we need branch farther than can be represented in the 16 bits of the conditional branch instruction. How can we rewrite the following instruction?
beq $s0, $s1, L1

```
        bne $S0, $S1, L2
        j       L1
L2:
```

13. (6%) Please write the procedure of multiplying 0011_2 by 0010_2 according the following algorithm. (錯 1 扣 1)



	Product	Multiplier	Multiplicand
	0000 0000	0011	0010
1:	0010 0000	0011	0010
2:	0001 0000	0011	0010
3:	0001 0000	0001	0010
1:	0011 0000	0001	0010
2:	0001 1000	0001	0010
3:	0001 1000	0000	0010
1:	0001 1000	0000	0010
2:	0000 1100	0000	0010
3:	0000 1100	0000	0010
1:	0000 1100	0000	0010
2:	0000 0110	0000	0010
3:	0000 0110	0000	0010
	0000 0110	0000	0010

14. (4%) Please define overflow and underflow. (各 2)

Overflow!

– A positive exponent becomes too large to fit in the exponent field

Underflow!

– A negative exponent becomes too large to fit in the exponent field

15. **(15%) In MIPS, there are five stages: instruction fetch**, Instruction Decode, Execute, Memory, and Write Result. Please describe the behaviour in the stages. (各 3)

Stage 1: instruction fetch
- No matter what the instruction, the 32-bit instruction word must first be fetched from memory (the cache-memory hierarchy)
- Also, this is where we increment PC
  (that is, PC = PC + 4, to point to the next instruction: byte addressing so + 4)

Stage 2: Instruction Decode
- upon fetching the instruction, we next gather data from the fields (*decode* all necessary instruction data)
- first, read the Opcode to determine instruction type and field lengths
- second, read in data from all necessary registers
  - for add, read two registers
  - for addi, read one register
  - for jal, no reads necessary

Stage 3: ALU (Arithmetic-Logic Unit)
- the real work of most instructions is done here: arithmetic (+, -, *, /), shifting, logic (&, |), comparisons (slt)
- what about loads and stores?
  - lw $t0, 40($t1)
  - the address we are accessing in memory = the value in $t1 + the value 40
  - so we do this addition in this stage

Stage 4: Memory Access
- actually only the load and store instructions do anything during this stage; the others remain idle
- since these instructions have a unique step, we need this extra stage to account for them
- as a result of the cache system, this stage is expected to be just as fast (on average) as the others

Stage 5: Register Write
- most instructions write the result of some computation into a register
- examples: arithmetic, logical, shifts, loads, slt
- what about stores, branches, jumps?
  - don’ t write anything into a register at the end
  - these remain idle during this fifth stage