Ch1. Instructions : Language of Computers

1. ISA — 軟、硬件間 interface；造計算机規格

為何要有 Alignment restriction?

beq R1, R2, L : MIPS
= cmp R1, R2 : ARM
  beq L
比較結果改變 conditional code reg 中的 flag field

和MIPS比較
- add r0, r1
- cmp

ARM IS

High-Level Language Program — Compiler — MIPS IS

計算机中從高階語言到執行流程.

Translation hierarchy 12.

Assembler / Ressembler 10. — Machine Code — Instruction format

Machine Code — Linker — Executable file — Loader — Put to memory to execute

Hardware Information

為何 Register 个數是 32 个?
何謂 Spilling Reg?

Instruction Set

3. Memory
- Endianers
- Alignment

2. Register
- MIPS processor
  - General Purpose Register
    - fix 0 \$k0~k1
    - fix assembler \$at
    - Procedure Call \$a0~a3, \$v0~v1, \$ra
    - Memory Management \$sp, \$fp, \$gp
    - Variable / Const \$t0~t7, \$s0~s7
  - Special Purpose Register
    - PC : 下一級執行指令位址 — 4x4 = 64
    - Hi, Lo : 乘 : Hi : 左半部 32 bit; Lo : 右半部 32 bit
      除 : Hi : 餘數; Lo : 商數 — 連結到 Hard-friendly 除法器
- Coprocessor 0
  - 處理 Exception (EPC, cause)
    - EPC : 儲存造成 exception 的指令位址, 用作恢復
    - Cause : 記錄 OS 決定 exception type
- Coprocessor 1
  - 浮點數運算

Instruction Format
- R type
- I type
- J type

Addressing Mode 4.
- Register (add / slt / and) : 運算元 (operand) 來自暫存器
- Immediate (addi / slti / andi) : Operand 夾帶在指令中的 16 bit 的 immediate field
- Base or displacement (lw / sw) : Operand 在 MEM 中, 位址用 base-displacement 得到
- PC-relative (beq / bne) : 跳躍目的位址為 PC 值 (PC+4) 加上 imm16
- Pseudo-direct (j / jal) : 跳躍目的位址為指令 26 bit const 和 PC 左邊 4 个 bit

PC-relative 和 pseudo-direct 跳躍範圍?
- PC-relative : PC 位址上下 $2^{-17} \sim 2^{17}-4$ byte
- Pseudo-direct : 一个 block 內, 一个 block 為 256MB (2^{28} byte), 看左邊 4 bits

5. Design Principle
- Simplicity...
- Make...
- Good...
- Smaller...

Type 6.
- CISC v.s. RISC
  - Instruction length
  - Instruction format
  - Addressing mode
  - Operand
  - Data in memory

CISC :
1. x86
2. AMD Opetron

為何現今 ISA 趨於逐漸從 CISC 轉向 RISC?
1. 程式碼長度
2. efficiency
3. 實現 pipeline
4. 取向

7. Style
- Accumulator
- M-M
- Stack
- load-store

8. Pseudo Instruction
- not
- move
- li
- bge
- bgt
- blt
- ble
- swap
- sgt

9. General Instruction
- Data Transfer
  - lw / sw
  - lh / sh
  - lb / sb
  - lhu / shu
  - lui / ori
- Operation
  - Arithmetic
    - Int
      - Signed
        - General
          - add
          - sub
          - mul
          - mult
          - div
          - slt
        - Immediate
          - addi
          - slti
      - Unsigned
        - General
          - addu
          - subu
          - multu
          - divu
          - sltu
        - Immediate
          - addiu
          - sltiu
    - Floating Point
  - Logic
    - shift
      - sll
      - srl
      - sra
    - Operation
      - General
        - and
        - or
        - nor
        - xor
      - Immediate

為何無 sub; 和 not 指令?
為何要有 pseudo-direct 指令?

- Flow Control
  - Instruction 間
    - Conditional
      - beq
      - bne
      - bgez
      - blez
    - unconditional
      - j
  - Procedure 間
    - jal
    - jr

Procedure Call 時需儲存哪些資訊避免錯誤?

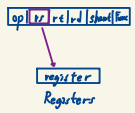
13. Program Tracing

1. ISA的定義：
   1. Hardware 和最底層軟件 (machine language) 間的 interface
   2. 相同ISA，不同 implementation 的 processor 可執行 相同軟件
   3. 不同implementation 會因 physical size, monetary cost 不同，性能不同
   4. ISA 影响 IC, CPI, clock cycle time

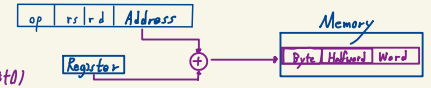
2. CPU & Register：
   I. Spilling register 观念：∵ Program中的变数多於Reg數目
   ∴ 將不常用变数放到MEM中，再用 load / store 指令做存取
   II. Reg 數目：
   MIPS中Reg 數目為 32个是基於 smaller is faster 設計原則
   Reg 越少，解碼時間越小，clock cycle time 越短，效能越高
   但对Complier来說 Reg 數越多：
   ① 分配变数给 Reg 迴圈变多，不用 spilling register，節省 load / store 時間
   ② 解決重覆命令的 antidependency，用 loop unrolling 時就可 Reg renaming
   III. $zero 的用途？實作方式？
   IV. $sp, $fp, $ra, $a0 ~ $a3 ?
   V. 為何 register set 要分為 GPR, floating register, SIMD register?
   VI. Register 大小會為 CPU一次存取資料大小 (字組大小)

3. Memory：
   I. Memory alignment：若字組大小為 4 byte，表示 CPU 一次可存取 4 byte 資料
   故資料的記憶体位址要為 4N (2進位，後面2个0)
   e.g. 0xF00ABCCC
   II. Endianess：big endian：位元組資料將 MSB 從 MEM 最低位址開始放起，e.g. MIPS
   little endian：位元組資料將 LSB 從 MEM 最低位址開始放起，e.g. AMD
   要什麼資料，以 big edian or little edian 放沒差？
   ① -1：1111…1
   ② IEEE 754 中的 0.0 = 000…0
   ③ C中的 null pointer = 00…0
   ④ 0：00…0
   ⑤ AB|CD|CD|AB

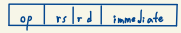
4. Addressing Mode：
   I. MIPS addressing mode
   ① Register：
   e.g. R-type
   add, sub



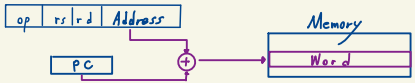
   ② Displacement：
   e.g. I-type
   lw $s0, 4($t0)



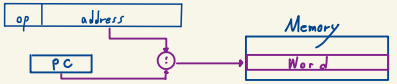
   ③ Immediate：
   e.g. I-type
   addi, slti;



   ④ PC-relative addressing
   e.g. I-type
   beq $s1, $s2, L1



   ⑤ Pseudo-direct addressing
   e.g. J-type
   j



   II. MIPS以外的 Addressing Mode
   ① 若有 immediate, direct, register, indexed 的 Addressing Mode，即夠處理大部份運算了
   ②

Addressing Mode	Example	Complier何時會使用
immediate	Add R4, #3	做常數運算時
displacement	Add R4, 100(R1)	存取 local variable 時
register indirect	Add R4, (R1) R4 ← R4 + MEM[R4+R1]	存取指標变數
scaled	Add R1, 100(R2)[R3] R1 ← R1 + MEM[100+R2+R3×d]	存取不同 array 相同 index

⇒ 為何左刁

11. High Level → MIPS IS ☆☆☆☆☆

1.

$s1 ← x $s2 ← y

x = y	beq $s1, $s2, lable
x ≠ y	bne $s1, $s2, lable
x < y	slt $t0, $s1, $s2 bne $t0, $0, lable
x ≤ y	slt $t0, $s2, $s1 beq $t0, $0, lable
x > y	slt $t0, $s2, $s1 bne $t0, $0, label
x ≥ y	slt $t0, $s1, $s2 beq $t0, $0, label

slt $t0, $s1, $s2
⇒ 若 s2 > s1 則 t0 = 1
s2 ≤ s1 t0 = 0

有 = ⇒ beq
沒有 = ⇒ bne

2.

$a0 ← A[ ] $s0 ← i

$t1 ← A[i]

```
sll  $t0, $s0, 2     // t0 = 4i
add  $t0, $t0, $a0   // address = a0 + 4i
lw   $t1, 0($t0)
```

3.

$s1 ← x $s2 ← y

```
if (x > y)
    statement
```

```
    slt  $t0, $s2, $s1
    beq  $t0, $0, end
    statement
end:
```

4.

$s1 ← x $s2 ← y

```
if (x ≥ y)
    statement_1
else
    statement_2
```

```
    slt  $t0, $s1, $s2
    bne  $t0, $0, else
    statement_1
    j    exit
else: statement_2
exit:
```

5.

$s1 ← x, $s2 ← y

```
while (x < y)
    statement
```

```
Loop: slt  $t0, $s1, $s2
      beq  $t0, $0, exit   // x ≥ y. exit
      statement
      j    Loop
exit:
```

6.

$a0 ← v[ ], $a1 ← k, $t0 ← temp

```
void swap(int v[], int k){
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

```
sll $t1, $a1, 2     // t1 = 4k
add $t1, $t1, $a0   // 算出 v[k] MEM 位址
lw  $t0, 0($t1)     // $t0 = v[k]
lw  $t2, 4($t1)     // $t2 = v[k+1]
sw  $t0, 4($t1)
sw  $t2, 0($t1)
```

7.

$s1 ← i

```
for (i=0; i<10; i++)
    statement
```

```
    add  $s1, $0, $0   // i 為 0
loop: slti $t0, $s1, 10  // s1 < 10, t1 = 1
    beq  $t0, $0, exit   // s1 ≥ 10, exit
    statement
    addi $s1, $s1, 1
    j    loop
exit:
```

```
    add  $s1, $0, $0
loop: statement
    addi $s1, $s1, 1
    slti $t1, $s1, 10   // s1<10, t1 = 1
    bne  $t1, $0, loop  // s1<10, loop
```

8.

$s1 ← a, $s2 ← b, $s3 ← A[ ], $s4 ← B[ ]

b = B[A[a+2]+1]

```
sll  $t0, $s1, 2
add  $t0, $t0, $s3
lw   $t1, 8($t0)
sll  $t1, $t1, 2
add  $t2, $t1, $s4
lw   $s2, 4($t2)
```

9.

$a0 ← a, $a1 ← b, $a3 ← c

```
int f(int a, int b, int c){
    return func(func(a,b),c);
}
```

```
f:  addi $sp, $sp, -8   // 開 stack 空間
    sw   $ra, 4($sp)  } 存 ra, s0
    sw   $s0, 0($sp)  }
    move $s0, $a2   // 存 c 到 $s0
    jal  func
    move $a0, $v0   // $a0 = func(a,b)
    move $a1, $s0   // 傳入 func 參數
    jal  func
    lw   $ra 4($sp) } 還原 ra, s0
    lw   $s0 0($sp) }
    addi $sp, $sp, 8   // 釋放 stack
    jr   $ra   // 返回 caller
```

f 內有 func(a,b)
⇒ func(func(a,b), c)
回傳結果

caller	callee
$ax	$ra
$tx	$sx

★★★★★ (Hard)

10.

$a0 ← n

```
int fact(int n){
  ①.
  if(n<1){
    ②.return 1;
  }
  else{
    ③. return (n*fact(n-1));
  }
}
```

```
fact: addi $sp, $sp, -8
      sw   $ra, 4($sp)
      sw   $a0, 0($sp)        } 存a0, ra
①.    slti $t0, $a0, 1 // $t0=1 if a0<1
      beq  $t0, $0, else
②.    addi $v0, $0, 1
      addi $sp, $sp, 8
      jr   $ra
else: addi $a0, $a0, -1 // 传入参数 n-1
③.    jal  fact
      lw   $a0, 0($sp)
      lw   $ra, 4($sp)        } 还原
      addi $sp, $sp, 8
      mul  $v0, $a0, $v0 // n*(n-1)!
      jr   $ra
```

步骤如下：

1. callee 存 ra, ax
2. 执行 if-else 判别
3. 自我时 e.g. 得参改变
4. 还原 ra, ax
5. 写入 return value 到 vx
6. 返回 caller

11

$a0 ← n

```
int sum(int n){
    if (n<1) return 0;
    else{
      return (n+sum(n-1));
    }
}
```

```
sum:  addi $sp, $sp, -8
      sw   $ra, 4($sp)
      sw   $a0, 0($sp)
      slti $t0, $a0, 1 // a0<1, to=1
      beq  $t0, $0, else
      add  $v0, $0, $0
      addi $sp, $sp, 8
      jr   $ra
else: addi $a0, $a0, -1
      jal  sum
      lw   $ra, 4($sp)
      lw   $a0, 0($sp)
      addi $sp, $sp, 8
      add  $v0, $a0, $v0
      jr   $ra
```

格式同上面的 fact
差别在返回值不同
callee 写入 V0 值不同