Computer Architectures
2. Instructions: Language of the Computer.13

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 13

- **Signed v.s. Unsigned**
- **Procedure Calling**



Signed v.s. Unsigned

- **Signed comparison: slt, slti**
- **Unsigned comparison: sltu, sltui**
- **Example**
  $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - **slt $t0, $s0, $s1 # signed**
    - **–1 < +1, so $t0 = 1**
  - **sltu $t0, $s0, $s1 # unsigned**
    - **+4,294,967,295 > +1, so $t0 = 0**



Outline

- …
- **Representing Instructions in the Computer**
- **Logical Operations**
- **Instructions for Making Decisions**
- **Supporting Procedures in Computer Hardware**



Procedure Calling

- **Steps required**
  1. **Place parameters in registers**
  2. **Transfer control to procedure**
  3. **Acquire storage for procedure**
  4. **Perform procedure's operations**
  5. **Place result in register for caller**
  6. **Return to place of call**



Review 13

- **Signed v.s. Unsigned**
- **Procedure Calling**

國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.14

Chun-Han Lin (林均翰)
CSIE, NTNU



Key Points 14

- **Register Usage**
- **Procedure Call Instructions**



Register Usage

- $a0 – $a3: arguments (registers 4 – 7)
- $v0, $v1: result values (registers 2 and 3)
- $t0 – $t9: temporary values
  - Can be overwritten by callee
- $s0 – $s7: saved variables
  - Must be saved/restored by callee
- $gp: global pointer for static data (register 28)
- $sp: stack pointer (register 29)
- $fp: frame pointer (register 30)
- $ra: return address (register 31)



Procedure Call Instructions

- **Procedure call: jump and link**

  jal ProcedureLabel

  - **Address of the following instruction put in $ra**
  - **Jump to target address**
- **Procedure return: jump register**

  jr $ra

  - **Copy $ra to program counter**
  - **Can also be used for computed jumps**
    - **E.g., for case/switch statements**



Review 14

- **Register Usage**
- **Procedure Call Instructions**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.15

Chun-Han Lin (林均翰)
CSIE, NTNU



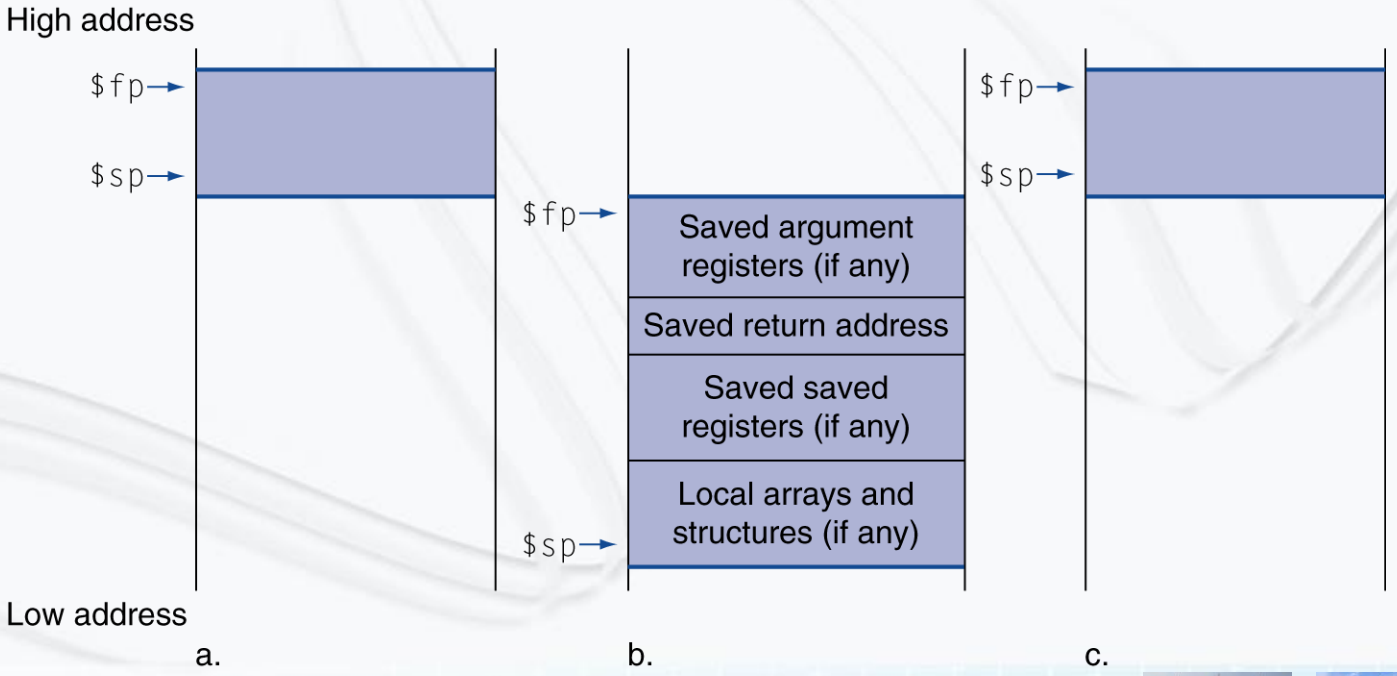
Key Points 15

- **Local Data on the Stack**



Local Data on the Stack

- **Local data is allocated by callee.**
  - **E.g., C automatic variables**
- **Procedure frame (activation record)**
  - **Used by some compilers to manage stack storage**



Review 15

- **Local Data on the Stack**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.16

Chun-Han Lin (林均翰)
CSIE, NTNU



Key Points 16

- **Leaf Procedure Example**



Leaf Procedure Example (1/2)

- C code

```
int leaf_example (int g, h, i, j){
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

  - Arguments g, …, j in $a0, …, $a3
  - f in $s0 (hence, need to save $s0 on stack)
  - Result in $v0



Leaf Procedure Example (2/2)

```
int leaf_example (int g, h, i, j){
        int f;
        f = (g + h) - (i + j);
        return f;
}
```

- MIPS code

leaf_example:	
addi $sp, $sp, -4 sw $s0, 0($sp)	Save $s0 on stack

add $t0, $a0, $a1
add $t1, $a2, $a3
sub $s0, $t0, $t1
	Procedure body

add $v0, $s0, $zero	Result
lw $s0, 0($sp) addi $sp, $sp, 4	Restore $s0
jr $ra	Return

Review 16

- **Leaf Procedure Example**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.17

Chun-Han Lin (林均翰)
CSIE, NTNU



Key Points 17

- **Non-Leaf Procedures**

- **Non-Leaf Procedure Example**



Non-Leaf Procedures

- **Procedures that call other procedures**

- **For nested call, caller needs to save on the stack …**
  - **Its return address**
  - **Any arguments and temporaries needed after the call**

- **Restore from the stack after the call**



Non-Leaf Procedure Example (1/2)

- C code

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

  - Argument n in $a0
  - Result in $v0



Non-Leaf Procedure Example (2/2)

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

- MIPS code

```
fact:
    addi $sp, $sp, -8     # adjust stack for 2 items
    sw   $ra, 4($sp)      # save return address, $ra
    sw   $a0, 0($sp)      # save 1 argument, $a0
    slti $t0, $a0, 1      # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1    # if so, result is 1
    addi $sp, $sp, 8      # pop 2 items from stack
    jr   $ra              # and return
L1: addi $a0, $a0, -1     # else decrement n
    jal  fact             # recursive call
    lw   $a0, 0($sp)      # restore original n
    lw   $ra, 4($sp)      # restore return address
    addi $sp, $sp, 8      # pop 2 items from stack
    mul  $v0, $a0, $v0    # multiply to get result
    jr   $ra              # and return
```




Review 17

- **Non-Leaf Procedures**
- **Non-Leaf Procedure Example**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.18

Chun-Han Lin (林均翰)
CSIE, NTNU



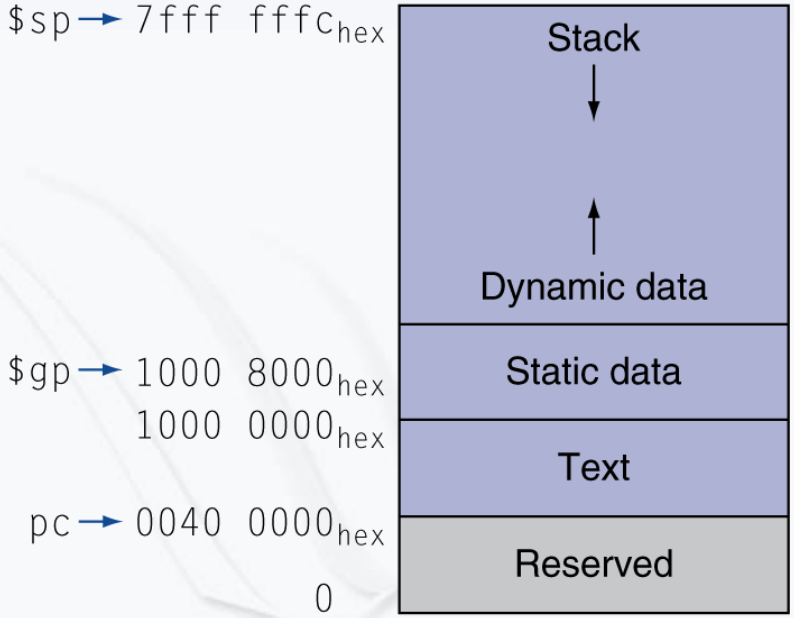
Key Points 18

- **Memory Layout**
- **Character Data**



Memory Layout

- Text: program code
- Static data: global variables
  - E.g., static variables in C, constant arrays and strings
  - $gp is initialized to address allowing ±offsets into this segment.
- Dynamic data: heap
  - E.g., malloc in C, new in Java
- Stack: automatic storage



Outline

- …
- **Logical Operations**
- **Instructions for Making Decisions**
- **Supporting Procedures in Computer Hardware**
- **Communicating with People**



Character Data

- **Byte-encoded character set**
  - **ASCII: 128 characters = 95 graphic + 33 control**
  - **Latin-1: 256 characters = ASCII + 96 more graphic characters**
- **Unicode: 32-bit character set**
  - **Used in Java, C++ wide characters, …**
  - **Most of world's alphabets, plus symbols**
  - **UTF-8, UTF-16: variable-length encodings**



Review 18

- **Memory Layout**
- **Character Data**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.19

Chun-Han Lin (林均翰)
CSIE, NTNU



Key Points 19

- **Byte/Halfword Operations**



Byte/Halfword Operations

- Could use bitwise operations
- MIPS byte/halfword load/store
  - String processing is a common case.
  - Sign extend to 32 bits in rt
    lb rt, offset(rs) lh rt, offset(rs)
  - Zero extend to 32 bits in rt
    lbu rt, offset(rs) lhu rt, offset(rs)
  - Store just rightmost byte/halfword
    sb rt, offset(rs) sh rt, offset(rs)



Review 19

- **Byte/Halfword Operations**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.20

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 20

- **String Copy Example**



String Copy Example (1/2)

- C code (naïve)
  - Null-terminated string

```
void strcpy (char x[], char y[]){
        int i;
        i = 0;
        while ((x[i]=y[i])!='\0')
                i += 1;
}
```

  - Addresses of x, y in $a0, $a1
  - i in $s0

String Copy Example (2/2)

```
void strcpy (char x[], char y[]){
        int i;
        i = 0;
        while ((x[i]=y[i])!='\0')
                i += 1;
}
```

- MIPS code

```
strcpy:
    addi $sp, $sp, -4  # adjust stack for 1 item
    sw   $s0, 0($sp)  # save $s0
    add  $s0, $zero, $zero  # i = 0
L1: add  $t1, $s0, $a1  # addr of y[i] in $t1
    lbu  $t2, 0($t1)  # $t2 = y[i]
    add  $t3, $s0, $a0  # addr of x[i] in $t3
    sb   $t2, 0($t3)  # x[i] = y[i]
    beq  $t2, $zero, L2  # exit loop if y[i] == 0
    addi $s0, $s0, 1  # i = i + 1
    j    L1  # next iteration of loop
L2: lw   $s0, 0($sp)  # restore saved $s0
    addi $sp, $sp, 4  # pop 1 item from stack
    jr   $ra  # and return
```




Review 20

- **String Copy Example**



Computer Architectures
2. Instructions: Language of the Computer.21

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 21

- **32-bit Constant**
- **Branch Addressing**



Outline

- …
- **Instructions for Making Decisions**
- **Supporting Procedures in Computer Hardware**
- **Communicating with People**
- **MIPS Addressing for 32-Bit Immediates and Addresses**



32-bit Constant

- Most constants are small.
- 16-bit immediate is sufficient.
- For an occasional 32-bit constant
  lui rt, constant
  - Copy 16-bit constant to left 16 bits of rt
  - Clear right 16 bits of rt to 0

```
lui $s0, 61
```

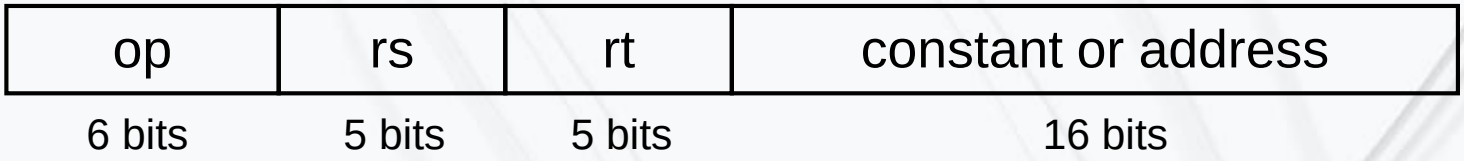
0000 0000 0111 1101	0000 0000 0000 0000

```
ori $s0, $s0, 2304
```

0000 0000 0111 1101	0000 1001 0000 0000

Branch Addressing

- Branch instructions specify …
  - Opcode, two registers, target address
- Most branch targets are near branch.
  - Forward or backward



- PC-relative addressing
  - Target address = PC + offset × 4
  - PC is already incremented by 4 by this time.



Review 21

- **32-bit Constant**
- **Branch Addressing**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.22

Chun-Han Lin (林均翰)
CSIE, NTNU



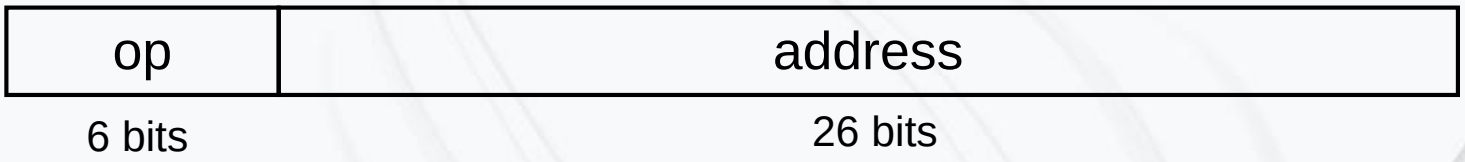
Key Points 22

- **Jump Addressing**
- **Target Addressing Example**



Jump Addressing

- **Jump (j and jal) targets could be anywhere in text segment.**
  - **Encode full address in an instruction**



- **(Pseudo)Direct jump addressing**
  - **Target address = PC31…28 : (address × 4)**

Target Addressing Example

- **Loop code from the earlier example**
  - **Assume Loop at location 80000**

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024						

Review 22

- **Jump Addressing**
- **Target Addressing Example**



國立臺灣師範大學
NATIONAL TAIWAN NORMAL UNIVERSITY

Computer Architectures
2. Instructions: Language of the Computer.23

Chun-Han Lin (林均翰)
CSIE, NTNU



Key Points 23

- **Branch Far Away**
- **Addressing Mode Summary**



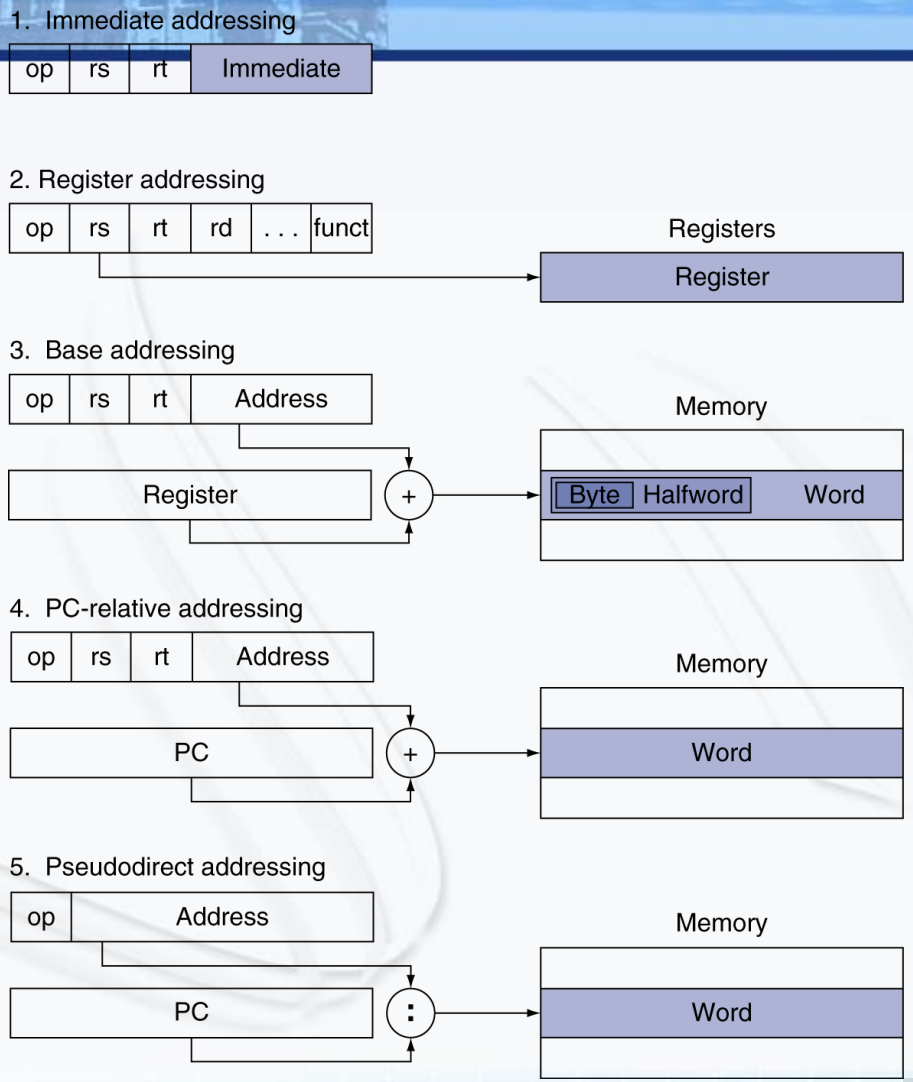
Branch Far Away

- If a branch target is too far to encode with 16-bit offset, assembler rewrites the code.

- Example

```
    beq $s0,$s1, L1  # 16 bits
                  ↓
    bne $s0,$s1, L2  # 16 bits
    j L1             # 26 bits
L2:      …
```

Addressing Mode Summary



Review 23

- **Branch Far Away**
- **Addressing Mode Summary**



Computer Architectures
2. Instructions: Language of the Computer.24

Chun-Han Lin (林均翰)

CSIE, NTNU



Key Points 24

- **Synchronization**
- **Synchronization in MIPS**



Outline

- …
- **Supporting Procedures in Computer Hardware**
- **Communicating with People**
- **MIPS Addressing for 32-Bit Immediates and Addresses**
- **Parallelism and Instructions: Synchronization**



Synchronization

- Two processors share an area of memory.
  - P1 writes, then P2 reads
  - Data race if P1 and P2 don't synchronize
    - Result depends of order of accesses.
- Hardware support is required.
  - Atomic read/write memory operation
  - No other access to a location is allowed between read and write.
- Could be a single instruction
  - E.g., atomic swap of register ↔ memory
- Or an atomic pair of instructions



Synchronization in MIPS

- **Load linked: ll rt, offset(rs)**
- **Store conditional: sc rt, offset(rs)**
  - **Succeed if the location is not changed since the ll**
    - **Returns 1 in rt**
  - **Fail if the location is changed**
    - **Returns 0 in rt**
- **Example: atomic swap $s4 with M[$s1] (to test/set lock variable)**

```
try: add $t0,$zero,$s4   #copy exchange value
     ll  $t1,0($s1)      #load linked
     sc  $t0,0($s1)      #store conditional
     beq $t0,$zero,try   #branch store fails
     add $s4,$zero,$t1   #put load value in $s4
```

Review 24

- **Synchronization**
- **Synchronization in MIPS**