CS 465 Lecture 3

MIPS ISA II: MIPS Encoding

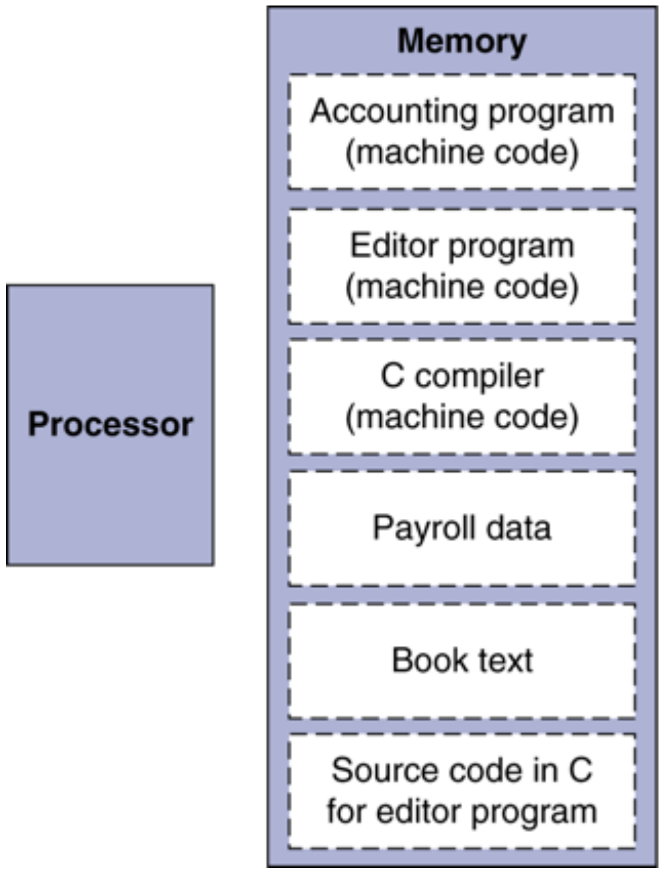
*Slides adapted from Computer Organization and Design by Patterson and Henessey

Road Map – MIPS ISA

- MIPS basic instructions
- MIPS instruction format ←
- Procedure calls
- Misc.

Stored Program Computers

The BIG Picture


Memory
Accounting program (machine code)
Editor program (machine code)
C compiler (machine code)
Payroll data
Book text
Source code in C for editor program
Processor

- Instructions represented in binary, just like data
- Instructions and data stored in memory
- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs

Representing Instructions

- Instructions are encoded in binary
  - Called machine code
- MIPS instructions: regularity is favored!
  - Encoded as 32-bit instruction words
  - Divided into fields
    - Each field encodes one thing: opcode, register, immediate …
  - MIPS defines only three basic types of instruction formats
    - Limited number of ways to divide/order fields

Instruction Formats

- I-format: immediate format
  - Instructions with immediate operand
    - Excluding shift instructions
  - Data transfer instructions (offset counts as an immediate)
  - Branches (beq and bne)
- J-format: jump format
  - j and jal (more details later)
- R-format: used for all other instructions

MIPS R-format Instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Instruction fields
  - op: operation code (opcode)
    - opcode=0 for all R-type instructions
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount
  - funct: function code (extends opcode)
    - Combined with opcode, exactly specifies the instruction

MIPS Opcode/Funct

Instruction	Opcode	Funct
add	00 0000	10 0000
sub	00 0000	10 0010
and	00 0000	10 0100
sll	00 0000	00 0000
srl	00 0000	00 0010
slt	00 0000	10 1010

All instructions in this table are R-type (Opcode = 000000)

R-format Example

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$0000\ 0010\ 0011\ 0010\ 0100\ 0000\ 0010\ 0000_2 = 02324020_{16}$

MIPS I-format Instructions

R-format	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Ideally, only one instruction format
  - Problem?
- *Design Principle 4:* Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

I-format	op	rs	rt	constant or address
	6 bits	5 bits	5 bits	16 bits

MIPS I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Immediate arithmetic and load/store instructions
  - Opcode: uniquely specifies an instruction
  - rs: source register number
  - rt: destination or (the other) source register number
- Immediate field
  - Used to specify immediates for instructions with a numerical constant operands: -2^{15} to $+2^{15} - 1$: `addi rt, rs, imm`
  - Used to specify address offset in data transfer instructions: lw, sw, etc: `lw rt, offset(rs)`
  - Used to specify branch address in bne and beq: `beq rs, rt, label`

I-format Example

- MIPS Instruction: addi $21,$22,-50
  - Encode for each field
    - opcode = 8 (look up in table in book)
    - rs = 22 (register containing operand)
    - rt = 21 (target register)
    - immediate = -50
  - Decimal number per field representation

8	22	21	-50

  - Binary number per field representation

001000	10110	10101	1111111111001110

    - Hexadecimal representation: 22D5 FFCE$_{hex}$

MIPS J-format Instructions

- J-format is used by MIPS jump instructions
  - j and jal
  - 6-bit opcode + 26-bit jump address

6 bits	26 bits
opcode	target address

  - Keep opcode field identical to R-format and I-format for consistency
  - Combine all other fields to make room for large target address
    - Goto statements and function calls tend to have larger offsets than branches and loops

Summary: MIPS Instruction Format

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

- Simplicity favors regularity
  - Fixed-length
  - Keeps formats as similar as possible
  - Benefit: simplified fetch/decoding logic

MIPS Decoding Example

- The six machine language instructions in binary:

```
R   00000000000000000001000000100101
R   00000000000000001010100000000101010
I   00010001000000000000000000000011
R   00000000010001000001000000100000
I   00100000101001011111111111111111
J   00001000000100000000000000000001
```

- First step: identify opcode and format

R	0	rs	rt	rd	shamt	funct
I	1, 4-63	rs	rt	immediate		
J	2 or 3	target address				

MIPS Decoding Example

- Now fields can be separated based on format / opcode:

R	0	0	0	2	0	37
R	0	0	5	8	0	42
I	4	8	0	+3		
R	0	2	4	2	0	32
I	8	5	5	-1		
J	2	1,048,577				

- Next step: translate (disassemble) to MIPS instructions

MIPS Decoding Example

op	rs	rt	rd	shamt	funct
0	0	0	2	0	37

- (op, funct) ➔ opcode
  - (0, 37)➔ or
- Register fields (rs/rt/rd): numeric value ➔ register number
  - Destination register: $2
  - Both operand registers: $0
- Assembly: or $2, $0, $0

MIPS Decoding Example

op	rs	rt	immediate
8	5	5	-1

- (op, funct) ➔ opcode
  - (8, -)➔ addi
- Register fields: numeric value ➔ register number
  - Destination register: $5
  - Operand register: $5
- Immediate field: numeric value ➔ constant
- Assembly: addi $5, $5, -1

MIPS Opcode/Funct

Instruction	Opcode	Funct
add	00 0000	10 0000 (32)
addi	00 1000 (8)	-----
or	00 0000	10 0100 (37)
slt	00 0000	10 1010 (42)
beq	00 0100 (4)	-----
j	00 0010 (2)	-----

MIPS Decoding Example

- Current MIPS

Address   Assembly instructions

```
        0x00400000    or    $2,$0,$0
        0x00400004    slt   $8,$0,$5
        0x00400008    beq   $8,$0,3
        0x0040000c    add   $2,$2,$4
        0x00400010    addi  $5,$5,-1
        0x00400014    j     0x100001
```

  - Will come back to this later
- Steps
  - Identify opcode and format
  - Fields separated based on opcode/format
  - Translate (disassemble) to MIPS

Recap: I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- Immediate arithmetic and load/store instructions
- immediate field
  - Used to specify immediates for instructions with a numerical constant operands: -2^{15} to $+2^{15} - 1$
  - Used to specify address offset in data transfer instructions: lw, sw, etc.
  - Used to specify branch address in bne and beq
- What about large immediates? Jumping far away?

Large Immediates

- New instruction:

```
    lui   $rt, immediate
```

  - Load Upper Immediate
  - Takes 16-bit immediate and puts these bits in the upper half (high order half) of the specified register; lower half is set to 0s
  - Example:
    - Want to do: `addi $t0,$t0, 0xABABCDCD`
    - Need to write a sequence instead:

```
lui    $at,0xABAB
ori    $at,$at,0xCDCD
add    $t0,$t0,$at
```

Branch Addressing

- Branch instructions specify
  - Opcode, two registers, target address
- Can branch forward or backward
  - Potential problem?
  - How large is the address space?
    - What is the length of byte addresses (PC register)?

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Immediate in Conditional Branches

- Observation:
  - Branch not very far
  - Why? Statistical support?
- Strategy: PC-relative addressing
  - No absolute address ➔
    - Only need to specify the difference between the branch target and the current instruction address
  - Instructions are words – must be aligned ➔
    - Specify immediate offset in words instead of bytes

Branch Address Calculation

- Calculation:
  - If we do not take the branch:

    $$PC_{new} = PC + 4$$

    - PC+4 = byte address of next instruction
  - If we do take the branch:

    $$PC_{new} = (PC + 4) + (immediate * 4)$$

- Observations
  - Immediate field specifies the number of words to jump, which is simply the number of instructions to jump
  - Immediate field can be positive or negative
  - Due to hardware, add immediate to (PC+4), not to PC; will be clearer why later in course

Decoding Example Revisited

- Current MIPS

Address	Assembly instructions	
0x00400000	or	$2,$0,$0
0x00400004	slt	$8,$0,$5
0x00400008	beq	$8,$0,3
0x0040000c	add	$2,$2,$4
0x00400010	addi	$5,$5,-1
0x00400014	j	0x100001
0x00400018		

Where is the branch target?

Encoding Example

- Current MIPS

```
Address                 Assembly instructions
      0x00400000         or     $2,$0,$0
      0x00400004         slt    $8,$0,$5
      0x00400008         beq    $8,$0,Next
      0x0040000c         add    $2,$2,$4
      0x00400010         addi   $5,$5,-1
      0x00400014         j      0x100001
      0x00400018 Next:
```

What is the lowest 16-bit?

Encoding Branch Offset

- Example
  - Difference between current addr and target is 16
    - 0x00400018 – 0x00400008 = 0x10 = 16
  - Target = PC + 16
  - Target = (PC + 4) + immediate * 4

  - ➔16 = 4 + immediate * 4
  - ➔immediate = 3

Branching Far Away

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

```
          beq $s0,$s1, L1
                  ↓
          bne $s0,$s1, L2
          j L1
    L2:   …
```

Jump Addressing

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction
  - Specify address in words ➔ can cover 28-bit address space
  - Where do we get the other 4 bits?
- Take the 4 highest order bits from PC+4
  - Not perfect, but adequate 99.9999…% of the time, since programs aren't that long
  - Problematic only if straddle a 256 MB boundary
- Backup plan: use jr instruction

op	address
6 bits	26 bits

Jumping Address

- Target address calculation
  - PC = PC + 4   #default updating
  - PC_{new} = { PC[31..28] || target address || 00 }
    = { PC[31..28] || target address <<2 }
    - || denotes concatenation
    - Understand where each part came from!
  - { 4 bits || 26 bits || 2 bits } = 32 bit address

Decoding Example Revisited

- Current MIPS

Address	Assembly instructions
0x00400000	or $2,$0,$0
0x00400004	slt $8,$0,$5
0x00400008	beq $8,$0,3
0x0040000c	add $2,$2,$4
0x00400010	addi $5,$5,-1
0x00400014	j 0x100001

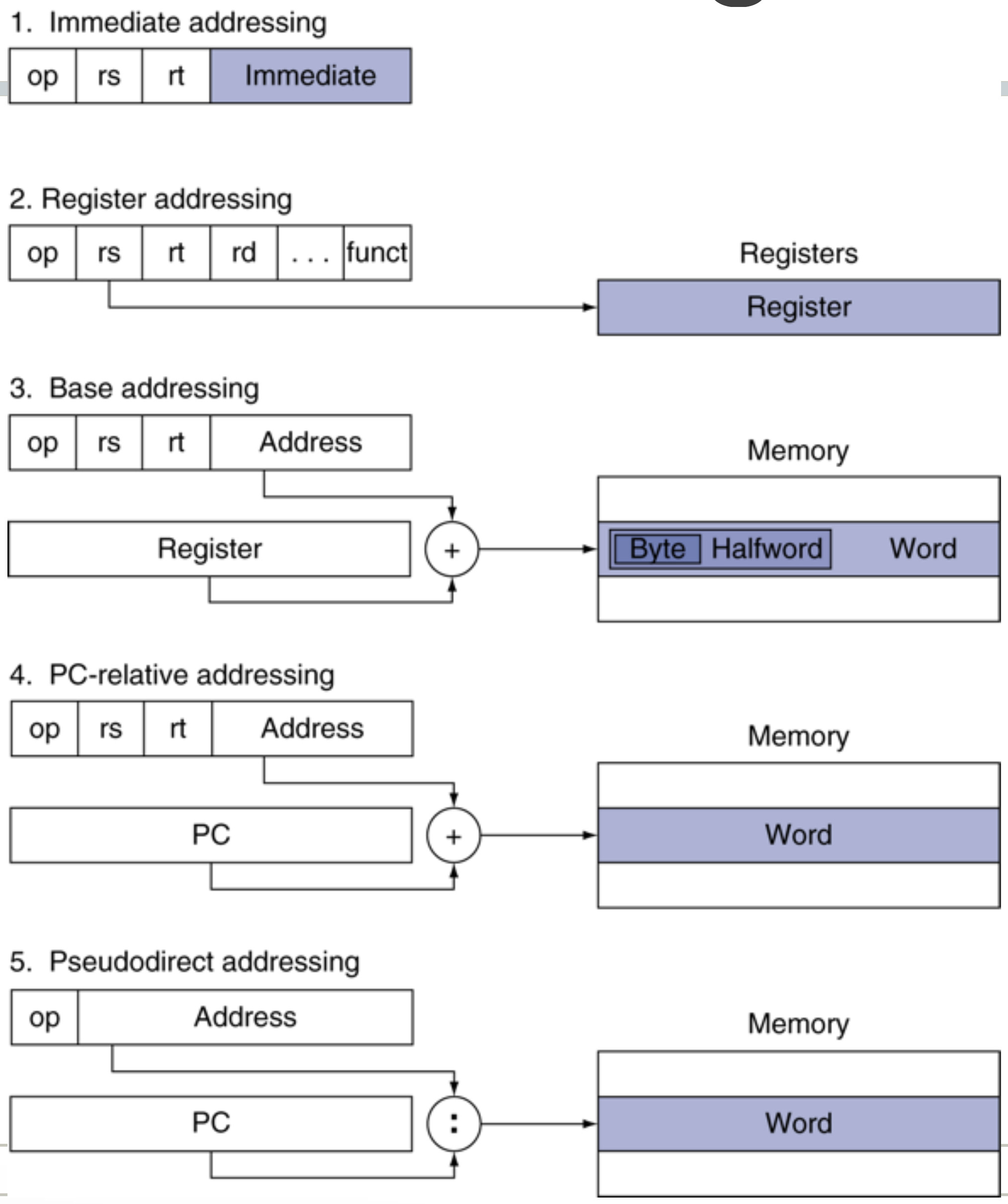
Where is the jump target?

Decoding Example

- MIPS Assembly:

```
     or     $v0,$0,$0
Loop:slt $t0,$0,$a1
     beq    $t0,$0,Exit
     add    $v0,$v0,$a0
     addi   $a1,$a1,-1
     j      Loop
Exit:
```

MIPS Addressing Modes



Summary

- MIPS instruction format
  - R/I/J format
  - Instruction encoding/decoding
- Encoding issues & MIPS solutions
  - Large constants
  - Branching addresses
  - Jumping addresses