CSSE232
Computer Architecture

ISAs

Reading

- For today:
  - Sections 2.1-2.5

- For next time:
  - Appendix B 9-10
  - I strongly encourage you to read this tonight!
  - Bring computer for lab!

Outline

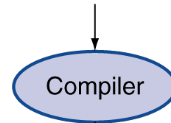
- Introduction to ISAs
- MIPS
  - Registers
  - Register operands
  - Memory operands
  - Representing instructions

Outline

- Review: Process of generating an executable
- Review: Introduction to ISAs
- MIPS
  - Registers
  - Register operands
  - Memory operands
  - Representing instructions

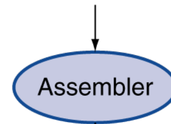
High-level
language
program
(in C)

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



Assembly
language
program
(for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



Binary machine
language
program
(for MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instruction Set

- The set of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Allowed simplified implementations
- Many modern computers also have simple instruction sets

The MIPS Instruction Set

- Used as the example throughout the book
- Typical of many modern ISAs
  - See MIPS Reference Data tear-out card
  - Appendixes B and E

- We will look at
  - Assembly format
  - Registers and memory use
  - Machine code format

Assembler

- a = b + c;

- d = e – f;

Assembler

- a = b + c;
  - add a, b, c

- d = e – f;
  - sub d, e, f

Assembler

- a = b + c;
  - add a, b, c

- d = e – f;
  - sub d, e, f

- What are a, b, … f?

Registers

- Hardware on CPU
- Very fast small memories

- MIPS has 32
- We will denote registers with '$'
  - So, $0 through $31
  - We will give names to them later

Assembler

- a = b + c;
  - add a, b, c
  - add $1, $2, $3
- d = e – f;
  - sub d, e, f
  - sub $4, $5, $6

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c  # a gets b + c
```

- All arithmetic operations have this form
- Design principle?

Example #1

- C code:

```
f = (g + h) - (i + j);
```

- Compiled MIPS code:

```
add $t1, $t1, $t2 #temp t1=g+h
add $t3, $t3, $t4 #temp t3=i+j
sub $t0, $t1, $t3 #f=t1-t3
```

Register Operands

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a "word"
- Assembler names
  - $t0, $t1, …, $t9 for temporary values
  - $s0, $s1, …, $s7 for saved variables

Register Usage

Register #	Register Name	Description
0	zero	Hardwired to zero
1	at	For assembler use
2	v0	Return values from procedure calls
3	v1	
4	a0	Arguments passed to procedure calls
5	a1	
6	a2	
7	a3	

Register Usage

Register #	Register Name	Description
8	t0	Temporary values (caller saves)
9	t1	
10	t2	
11	t3	
12	t4	
13	t5	
14	t6	
15	t7	

Register Usage

Register #	Register Name	Description
16	s0	Save values (callee saves)
17	s1	
18	s2	
19	s3	
20	s4	
21	s5	
22	s6	
23	s7	

Register Usage

Register #	Register Name	Description
24	t8	Temporary values caller saves
25	t9	
26	k0	Reserved for OS Kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	Frame pointer
31	ra	Return address

Example #2

- C code:

```
f = (g + h) - (i + j);
```

f, g, h, i, j in $s0, $s1, $s2, $s3, $s4

- Compiled MIPS code:

```
add $t0, $s1, $s2  #temp t0 = g + h
add $t1, $s3, $s4  #temp t1 = i + j
sub $s0, $t0, $t1  #f = t0 - t1
```

Memory Operands

- Programs often store lots of data
  - 32 general registers, only 18 for user
  - Need another place to store data
- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory

Memory Operands

- Memory is **byte addressed**
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a **multiple of 4**
  - We will be using words in this class!
- MIPS is **Big Endian**
  - Most-significant byte at least address of a word
  - *cf.* Little Endian: least-significant byte at least address

Example #3

- C code:

```
g = h + A[8];
```

  - g in $s1, h in $s2, base address of A in $s3
- Compiled MIPS code:
  - Index 8 requires offset of 32
    - 4 bytes per word

```
lw  $t0, 32($s3)     # load word
add $s1, $s2, $t0
```

Example #4

- C code:

```
A[12] = h + A[8];
```

  - h in $s2, base address of A in $s3

- Compiled MIPS code:
  - Index 8 requires offset of 32

```
lw  $t0, 32($s3)      #load word
add $t0, $s2, $t0
sw  $t0, 48($s3)      #store word
```

Registers vs. Memory

- Registers are **faster** to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler use registers for variables as much as possible
  - Only spill to memory for **less frequently** used variables
  - Register optimization is important!

Registers vs. Memory

- Quiz #3
  - Why not keep all values in registers or memory?

Need to get values…

- Code: i = i+1
- How to do in MIPS?

Need to get values…

- Code: i = i+1
- How to do in MIPS?

- Need to get values into registers, some how…

Immediate Operands

- Very common to use constants in programs
  - Constant value can be supplied with instruction
  - Called: immediate value

- Example of constant data specified in an instruction
  `addi $s3, $s3, 4`

- No subtract immediate instruction!
  - Just use a negative constant
    `addi $s2, $s1, -1`

The Constant Zero

- MIPS register 0 ($zero) is the constant 0
  - Cannot be overwritten
- Useful for common operations
  - E.g., move between registers

```
add $t2, $s1, $zero
```

Representing Instructions

- How?

Representing Instructions

- Instructions are encoded in binary
  - Called machine code
- MIPS instructions
  - Encoded as 32-bit instruction **words**
  - Small number of formats encoding operation code (opcode), register numbers, …
- Register numbers
  - $t0 – $t7 are registers 8 – 15
  - $t8 – $t9 are registers 24 – 25
  - $s0 – $s7 are registers 16 – 23

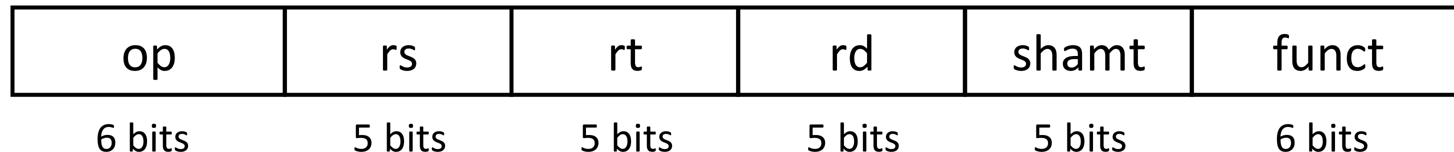
Instruction types

- 3 types of instructions in MIPS
  - R-types
  - I-types
  - J-types

- Each type has a different format
- All are 32 bits long

MIPS R-format Instructions

- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount (00000 for now)
  - funct: function code (extends opcode)



R-format Example

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

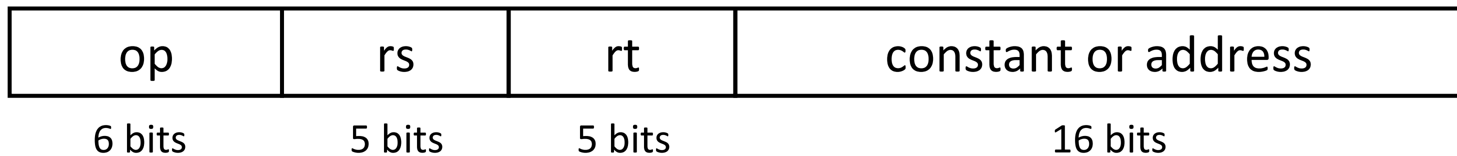
```
add $t0, $s1, $s2
```

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

$00000010001100100100000000100000_2 = 02324020_{16}$

MIPS I-format Instructions

- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - If address: offset added to base address in rs



I-format Example

- addi \$s0, \$s1, 2

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

op	\$s1	\$s0	0x2

001000	10001	10000	0000 0000 0000 0010

One more type: J types

- Take a look at your green sheet
- Jump (j) and Jump and Link (jal)
  - Opcode (*j* = 2, *jal* = 3)
  - 26 bit address
- What about *jr*?
  - Its an R-type instruction

Recap

- Process of generating an executable
- Introduction to ISA
- MIPS
  - Registers
  - Register operands
  - Memory operands
  - Representing instructions

Recap

- Introduction to ISA
- MIPS
  - Registers
  - Register operands
  - Memory operands
  - Representing instructions

MIPS reference

- Green sheet
- Page 135
- Appendix B-50