

Machine Language: the Software/Hardware Interface

◆ Last Time

- Summarizing performance (AM, WAM, GM)
- Benchmark Suites

◆ This Time

- Quiz
- Basics of Machine Language
- Instructions, format&sequence
- Assembly directives

◆ Announcements

- Read P&H Chapter 3.1-3.4
- Homework #1 is due Monday, 1/24 at the beginning of section (no late homeworks accepted)

CS 141 Chien

1

January 20, 1999

Basic Machine Operations

◆ Operations + Data = computation



◆ Sequences of operations -> a computation

◆ How to specify operations?

◆ How to specify the operands? (data)

CS 141 Chien

2

January 20, 1999

Assembly Language

- ◆ Symbolic form of machine language (readable)

- ◆ Example: (a+b+c+d+e)

```
add    a, a, b
add    a, a, c
add    a, a, d
add    a, a, e
```

- ◆ Three address operations - 2 in, 1 out
- ◆ Read operands, operate, write result, next instruction
- ◆ If in=out, read in, operate, write out

What operations?

- ◆ Each machine has a fixed set of operations.
- ◆ Requirements of hardware implementation
 - small fixed set
 - fixed number of operands
 - can be implemented to run fast
- ◆ Instructions based on common operations for workload
- ◆ Instructions based on what programmers and compilers can easily use
- ◆ => regular, simple instructions

Typical Basic Operations

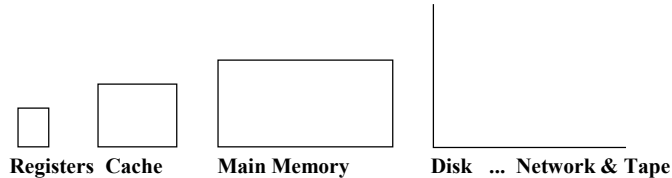
- ◆ **Arithmetic:**
 - add and sub
 - multiply and divide
 - moves
 - floating point operations
- ◆ **Logical: and, or, shift, etc.**
- ◆ **Memory operations (the next topic)**
- ◆ **Control flow operations (next lecture topic)**

CS 141 Chien

5

January 20, 1999

How to specify operands? (data)



- ◆ **Computers have memory hierarchies**
- ◆ **Smaller is faster (Einstein's speed limit!)**
- ◆ **Explicit memories allow clever management (programmer, compiler)**

	<u># avail</u>	<u>Specifier</u>	
Registers	32	\$n	\$1, \$4, \$17, etc.
Memory	1B	XX	1234, 5000, 100099, etc.

=> Operands that appear as source and destination of the instructions

CS 141 Chien

6

January 20, 1999

Specifying data (cont.)

- ◆ **Registers (32) -> 5 bits to encode**

- 32 bits each (word size)

- ◆ **Memory (1B) -> 30 bits to encode**

- variable size quantities
- Very large memories? 64GB – 1Terabyte?

- ◆ **How to resolve this?**

MIPS Instruction Set

=> only registers as operands for regular instructions

=> Addresses used only in LOAD and STORE operations

Idea: separate computation (register to register) from memory traffic (LOAD/STORE)

“Load/Store Architecture”

CS 141 Chien

7

January 20, 1999

Specifying data (cont.)

- ◆ **LOAD instructions**

```
lw    $8, 0($2)           # $8 <- Memory[0+$2]
```

- ◆ **STORE instructions**

```
sw    $5, 4($2)          # Memory[4+$2] <- $5
```

Idea: Load data into registers, compute on it,
eventually write it back into the memory.

CS 141 Chien

8

January 20, 1999

Operand Types

- ◆ Several operand types
 - Register
 - Memory
 - Instruction constant (immediate)
 - » addi, lui, etc.
 - » offsets in addressing mode 23(\$3)
- ◆ Values allowed depends on the bits available in the instruction and the width of the datapath

Example: Assembly programming

- ◆ Implementing an array computation
- ◆ $A[i+1] = c + A[i]$;
- ◆ i in \$20, c in \$19, A_{start} is address of start of array

```
lw    $2, Astart($20)    # $20 holds i
add   $3,$2,$19          # c is in $19
addi  $21, $20, 1        # put (i+1) in $21
sw    $3, Astart($21)    # store result in A[i+1]
```

=> Load, compute, store
=> separation of memory operations and compute operations

Memory addressing

- ◆ **Data comes in different sizes**
 - word: 32 bits “natural unit”
 - halfword: 16 bits
 - bytes: 8 bits
- ◆ **Registers are words, 32 bits each**
- ◆ **lw, sw are word operations**
- ◆ **Addresses correspond to bytes (for MIPS)**

n:				
n+4:				
n+8:				
n+12:				

One address per byte.
Word addresses are spaced by 4
Halfword addresses by 2
Alignment and non-aligned

Assembly and Machine Language

- ◆ **Assembly: symbolic notation, readability**
 - ◆ **Machine: actual execution format**
 - ◆ **all symbols -> encoded patterns of 1's and 0's**
 - ◆ **Instruction formats -- driven by demands of high speed implementation:**
 - regular structure, fields in same places
 - few fixed formats (match to fixed hardware decoders)
 - determines/determined by: # of Regs, constant sizes, etc.
- => How many instruction formats? 3 in MIPS, we'll look at 2 now.

Machine Instruction Formats

- ◆ All instructions are 32 bits (1 word)
- ◆ R format -- arithmetic and logic instructions
- ◆ I format -- transfer and branch instructions
- ◆ J format -- jump instructions

=> We'll look at R and I for now.

R format Instructions

- ◆ Arithmetic and Logical Instructions

op(6)	rs(5)	rt(5)	rd(5)	shamt(5)	funct(6)
-------	-------	-------	-------	----------	----------

op -- operation of the instructions
rs -- first source register number
rt -- second source register
rd -- destination register
shamt -- shift amount (later)
funct -- additional specification of operation

=> fixed field placement and width for all instructions in this format

=> some op bits are used to specify the instruction format

I format Instructions

- ◆ Load/Store and Branch Instructions
- ◆ Contains a large constant field (needed)

op(6)	rs(5)	rt(5)	address(16)
-------	-------	-------	-------------

op -- operation code
 rs -- register source
 rt -- used as destination register in this format
 address -- 16-bit constant for addressing

- => instructions in this class use at most two registers
- => this is how to get constants into the registers

CS 141 Chien

15

January 20, 1999

Instruction Formats: An Example

- ◆ Assembly and Instruction format correspondence

lw	\$2, Astart(\$20)	lw	20	2	Astart
add	\$3, \$2, \$19	add	2	19	3 add
addi	\$21, \$20, 1	addi	20	21	1
sw	\$3, Astart(\$21)	sw	21	3	Astart

- => Mapping is straightforward
- => addi uses the I format, insts with "immediates" do
- => Generally one to one correspondence
- => Straightforward translation

CS 141 Chien

16

January 20, 1999

Summary

- ◆ **Machine language is software hardware interface**
- ◆ **Instructions are how we describe a computation -- operations and data**
- ◆ **Memory types: registers, memory, addressing (bytes)**
- ◆ **Machine instructions encoded into a few fixed formats, supporting efficient execution**
- ◆ **NEXT TIME: Control flow (conditional execution)**