Computer Organization

605.204

Module Three

Part Three

Language for the People

Module Three

- Part Three
- In this presentation, we are going to talk about :
- Assembly Language
- The first set of MIPS assembly language instructions

Previously

- Previously we talked about:
- Language for the Machine
- Now: A Language for the People

People Language

- People read and write ‘natural’ languages
  - English
  - Spanish
  - Chinese
- People use words, phrases, nouns, pronouns, and verbs
- Natural languages are ambiguous (English is the worst)
  - “The chicken is ready to eat.”

People Language

- People prefer to write computer programs in languages that most easily allow them to describe the problems they need to solve.
- High level programming languages were developed for this very reason.
- But …first …
- People understand letter patterns better than number patterns

00	12	28
LDA	SUB	COMP

Hardware Assembly Language

Symbolic language,

- almost 1 to 1 with the machine language.

- Fixed Format Fixed or Variable Length

- One instruction per line

Label: Operation Fixed number of operands #Comments

```
loop:   LW      $v1, 0($a0)          # get the next source word
```

100011 00100 00010 0000000000000000

Who Uses Assembly Language

- The machine designer
  - Must implement and trade off instruction functionality
- The compiler writer
  - Must generate machine language from a High Level Languages
- The writer of time or space critical code
  - Performance goals may force program-specific optimizations of the assembly language
- Special purpose or imbedded processor programmers
  - Special functions and heavy dependence on unique devices can make High Level Languages useless
- Students

MIPS Design Overview

All processing operands are in registers
    to promote speed

Load and Store access to memory
    load operand data value into register from memory
    store operand data value from register to memory

Design goals: **Maximize performance,**
Minimize cost,
and Reduce design time

Operands in Registers

- To allow the MIPS processor to run faster
- MIPS Registers are 32 bits in size
  - Integer values are +/- 2 billion
  - Floating point values are +/- 1.70*10^38
  - 4 ASCII characters
- MIPS has 32 registers

Load and Store

- Move operand values from and to the main memory
- LOAD - copy value from memory to a register
  - LW – Load Word - copy all 32 bits to a register
- STORE - copy value to memory from a register
  - SW – Store Word - copy all 32 bits to memory

MIPS Assembly Instructions

- Language of this Machine
- More primitive than higher level languages
  No sophisticated control flow: Branch and Jump only
- Very restrictive
  MIPS Arithmetic Instructions
  ADD $t3, $t1, $t2 ($t3 = $t1 + $t2)
- We'll be working with the MIPS instruction set architecture
  - similar to other architectures developed since the 1980's
  - used by NEC, Nintendo, Silicon Graphics, Sony

Design goals: **Maximize performance, Minimize cost, and Reduce design time**

MIPS Assembly Language

- Assembly language provides convenient symbolic representation
  - much easier than writing down numbers
  - destination first
- Machine language is the underlying reality
- Assembly language can provide 'pseudo-instructions'
  - "**`move $t0, $t1`**" exists only in Assembler
  - would be implemented using "**`add $t0,$t1,$zero`**"
- When considering performance you should count real instructions

MIPS Assembly Arithmetic

- Instructions have 3 operands.
- Operand values are in registers.
- Operand order is fixed - destination first

Example:

C code: A = B + C

MIPS code: **`add $s0, $s1, $s2`**

A B C

(registers associated with variables by the compiler)

MIPS Assembly Arithmetic

- Design Principle: **Simplicity favors regularity**.
- Of course this complicates some things...

C code: A = B + C + D;
E = F - A;

MIPS code:

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

- Operands must be in registers, only 32 registers provided
- Design Principle: **Smaller is faster**.

Load and store Instructions

- Example:

C code: A[8] = h + A[8];

MIPS code:

```
lw  $t0, 32($s3) ; content of  A[8]
add $t0, $s2, $t0; add h
sw  $t0, 32($s3) ; save to A[8]
```

- Machine process: Memory address = rs + 16 bit value

Load and store Instructions

- Consider the load-word and store-word instructions,
  - What would the regularity principle have us do?
  - Design principle: **Good design demands a compromise**
- Example:

lw $s1, 100($s2)	$s1 = Memory[$s2+100]
sw $s1, 100($s2)	Memory[$s2+100] = $s1

- Where's the compromise?

  Store word has destination last.

At this point:

- MIPS
  - Loading words but addressing bytes
  - Arithmetic operands in registers only

Instruction	Meaning
`add $s1, $s2, $s3`	$s1 = $s2 + $s3
`sub $s1, $s2, $s3`	$s1 = $s2 – $s3
`lw $s1, 100($s2)`	$s1 = Memory[$s2+100]
`sw $s1, 100($s2)`	Memory[$s2+100] = $s1

Summary

- People Languages
- MIPS Assembly Language
- All processing operands are in registers
- Load and Store access to memory

Next: More of the MIPS Assembly Language