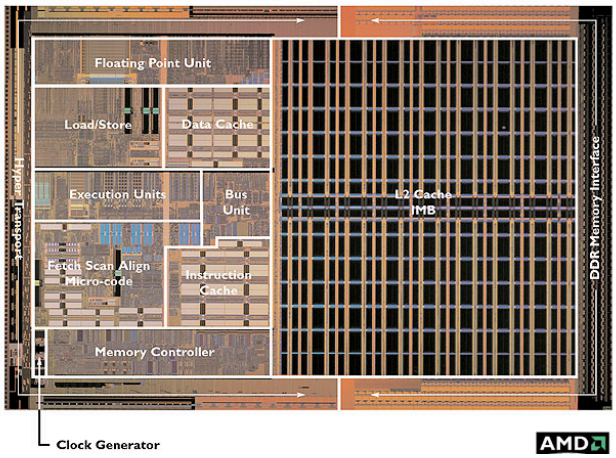
CSE378 Machine Organization



Larry Snyder

1

What is Computer Architecture?

Computer architecture is the study of building computer systems.

CSE378 is roughly split into three parts.

- The first third discusses instruction set architectures -- the bridge between hardware and software.
- Next, we introduce more advanced processor implementations. The focus is on pipelining, which is one of the most important ways to improve performance.
- Finally, we talk about memory systems, I/O, and how to connect it all together. Memory Processor Input/Output

2

Why should you care?

It's interesting
- ❖ You will learn how a processor actually works!

It will help you be a better programmer
- ❖ Understanding how your program is translated to assembly code lets you reason about correctness and performance.
- ❖ Demystify the seemingly arbitrary (*e.g.,* bus errors, segmentation faults).

Many cool jobs require an understanding of computer architecture.
- ❖ The cutting edge is often pushing computers to their limits.
- ❖ Supercomputing, games, portable devices, etc.

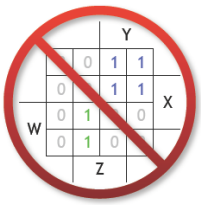
Computer architecture illustrates many fundamental ideas in computer science
- ❖ Abstraction, caching, and indirection are CS staples.

3

CSE370 vs CSE378

This class expands upon the computer architecture material from the last few weeks of CSE370, and we rely on many other ideas from CS370.
- ❖ Understanding binary, hexadecimal and twos-complement numbers is still important.
- ❖ Devices like multiplexers, registers and ALUs appear frequently. You should remember what they do, but not necessarily how they work. Finite state machines and sequential circuits will appear again.

We do *not* spend time with logic design topics like Karnaugh maps, Boolean algebra, latches and flip-flops.



4

Administrivia -- The People

Larry Snyder

Jacob Nelson

Jacob Nelson

Aaron Miller

Aaron Miller

LS office hours: M, Th @ 1:00-1:30 or by appointment to snyder AT cs

5

Administriva – The Course

The textbook provides the most comprehensive coverage (it's a beautiful textbook, easy to read & use)

- *Computer Organization and Design*, Patterson and Hennessy, 4th Edition

Lectures will present course material TAKE NOTES

Sections, you signed up for one; here's how they work

- We have EEB 025 (2:30-3:30) for "classroom work"
- We have CSE 003 Lab (2:30-5:30) for "lab work"
- Classroom meetings – Jacob will teach
- Lab meetings – Aaron or Jacob will be around
- Use lab time wisely, because they won't usually be around at other times

6

Administrivia – The Grading

Grading:

- Lab assignments: 25%
- Homeworks: 15%
- Midterm: 20%
- Final: 35%
- Participation: 5%

❖ Getting in touch with us:

- cs378@cs, cse378-tas@cs,
- course webpage:
- http://www.cs.washington.edu/education/courses/378/09au/

7

First Home Work

On Friday hand in (hardcopy) a page with the following information about yourself:

Name:

Preferred Email:

(Intended) Major(s):

Year:

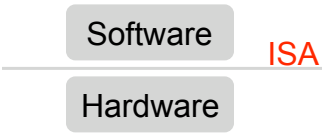
Hometown:

Photo:

An Interesting Fact About Yourself:

8

Instruction Set Architectures



Interface between hardware and software

Abstraction: hide HW complexity from the software through a set of simple operations and devices

`add, mul, and, lw, ...`

9

MIPS

In this class, we'll use the MIPS instruction set architecture (ISA) to illustrate concepts in assembly language and machine organization

- Of course, the concepts are not MIPS-specific
- MIPS is just convenient because it is real, yet simple (unlike x86)

The MIPS ISA is still used in many places today. Primarily in embedded systems, like:

- Various routers from Cisco
- Game machines like the Nintendo 64 and Playstation

10

From C to Machine Language

High-level language (C) **a = b + c;**

Assembly Language (MIPS) **add $16, $17, $18**

Binary Machine Language (MIPS) **01010111010101101...**

11

A Seemingly Useless Task

You must become "fluent" in MIPS assembly:

- Translate from C to MIPS and MIPS to C
- Example problem: Write a recursive function Here is a function pow that takes two arguments (n and m, both 32-bit numbers) and returns n^m (i.e., n raised to the mth power).

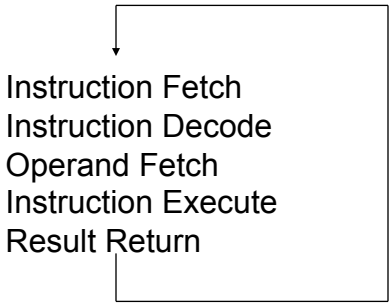
```
int pow(int n, int m) {
   if (m == 1)
      return n;
   return n * pow(n, m-1);
}
```

- Translate this into a MIPS assembly language function.

12

Instruction Execution Engines

As you know, computers are instruction execution engines that endlessly run the fetch/ execute cycle



This course explains in detail this logical process

13

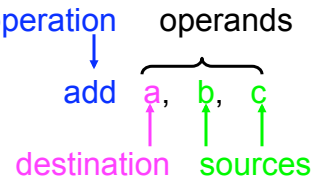
MIPS Register to Register Insts

MIPS is a register-to-register, or load/store, architecture.

- The destination and sources must all be registers.
- Special instructions, which we'll see soon, are needed to access main memory.

MIPS uses three-address instructions for data manipulation.

- Each ALU instruction contains a destination and two sources.
- For example, an addition instruction (a = b + c) has the form:



14

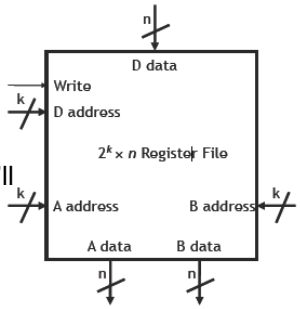
The MIPS Register File

MIPS processors have 32 registers, each of which holds a 32-bit value.

- Register addresses are 5 bits long.
- The data inputs and outputs are 32-bits wide.

More registers might seem better, but there's a limit to their value

- It's more expensive, because of both the registers themselves as well as the decoders and muxes needed to select individual registers.
- Instruction lengths may be affected, as we'll see in the future.



15

MIPS Register Names

MIPS register names begin with a $. There are two naming conventions:

- By number: $0 $1 $2 … $31
- By (mostly) two-character names, such as: $a0-$a3 $s0-$s7 $t0-$t9 $sp $ra

Not all of the registers are equivalent:

- E.g., register $0 or $zero always contains the value 0 (go ahead, try to change it)

Other registers have special uses, by convention:

- E.g., register $sp is used to hold the stack pointer

You have to be a little careful in picking registers for your programs … more about this later

16

Basic Arithmetic and Logic Operations

The basic integer arithmetic operations include the following:

add sub mul div

And here are a few logical operations:

and or xor

Remember that these all require three register operands; for example:

```
add $t0, $t1, $t2     # $t0 = $t1 + $t2
and $s1, $s1, $a0     # $s1 = $s1 ⋀ $a0
```

17

Larger Expressions

More complex arithmetic expressions may require multiple operations at the instruction set level.

$t0 = (t1 + t2) \times (t3 - t4)$

```
add $t0, $t1, $t2 # $t0 contains $t1 + $t2
sub $s0, $t3, $t4 # Temporary value $s0 = $t3 - $t4
mul $t0, $t0, $s0 # $t0 contains the final product
```

Temporary registers may be necessary, since each MIPS instructions can access only two source registers and one destination register

- In this example, we could re-use $t3 instead of introducing $s0.

- Much more detail in future lectures…

18

Immediate Operands

- The ALU instructions we've seen so far expect register operands. How does data get into registers in the first place?
- Some MIPS instructions allow a signed constant, or "immediate" value, for the second source instead of a register. For example, here is the immediate add instruction, addi:

  addi $t0, $t1, 4 # $t0 = $t1 + 4
- Immediate operands can be used in conjunction with the $zero register to write constants into registers:

  addi $t0, $0, 4 # $t0 = 4
- MIPS is still considered a load/store architecture, because arithmetic operands cannot be from arbitrary memory locations. They must either be registers or constants that are embedded in the instruction.

19

More Space!

Registers are fast and convenient, but we have only 32 of them, and each one is just 32-bits wide. That's not enough to hold data structures like large arrays.

We also can't access data elements that are wider than 32 bits.

We need to add some main memory to the system!

- ❖ RAM is cheaper and denser than registers, so we can add lots of it.
- ❖ But memory is also significantly slower, so registers should be used whenever possible. In the past, using registers wisely was the programmer's job.

For example, C has a keyword "register" that marks commonly-used variables which should be kept in the register file if possible. However, modern compilers do a pretty good job of using registers intelligently and minimizing RAM accesses.

20

How to Succeed in 378

Remember the big picture. What are we trying to accomplish, and why?

Read the textbook. It's clear, well-organized, and well-written. The diagrams can be complex, but are worth studying. Work through the examples and try some exercises on your own. Read the "Real Stuff" and "Historical Perspective" sections.

Talk to each other. You can learn a lot from other CSE378 students, both by asking and answering questions. Find some good partners for the homeworks/labs (but make sure you all understand what's going on).

Help us help you. Come to lectures, sections and office hours. Send email or post on the mailing list. Ask lots of questions! Check out the web page.

21