inst.eecs.berkeley.edu/~cs61c

UCB CS61C : Machine Structures

Lecture 13
MIPS Instruction Representation I

2010-02-19

Hello to
Hasitha Karunaratne
from Sri Lanka!

Lecturer SOE
Dan Garcia

TWO CHINESE SCHOOLS NAMED IN ATTACKS

Shanghai Jiaotong University and Lanxiang Vocational School have been traced to the online hacking attacks on Google and 20 other companies. When asked, a leading professor at SJU said *"I'm not surprised. Actually students hacking into foreign Web sites is quite normal."*

www.nytimes.com/2010/02/19/technology/19china.html

Review

- Register Conventions: Each register has a purpose and limits to its usage. Learn these and follow them, even if you're writing all the code yourself.

- Logical and Shift Instructions
  - Operate on bits individually, unlike arithmetic, which operate on entire word.
  - Use to isolate fields, either by masking or by shifting back and forth.
  - Use shift left logical, `sll`, for multiplication by powers of 2
  - Use shift right logical, `srl`, for division by powers of 2 of unsigned numbers (`unsigned int`)
  - Use shift right arithmetic, `sra`, for division by powers of 2 of signed numbers (`int`)

- New Instructions:
  `and, andi, or, ori, sll, srl, sra`

61C Levels of Representation (abstractions)

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g.,MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $t0, 0($s2)
lw    $t1, 4($s2)
sw    $t1, 0($s2)
sw    $t0, 4($s2)
```
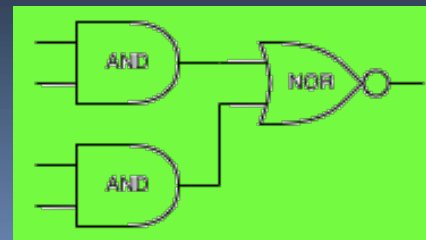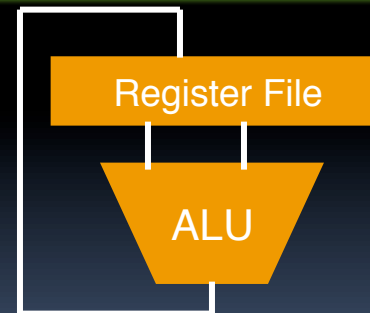
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Register File

ALU

AND
AND
NOR

Overview – Instruction Representation

- Big idea: stored program
  - consequences of stored program
- Instructions as numbers
- Instruction encoding
- MIPS instruction format for Add instructions
- MIPS instruction format for Immediate, Data transfer instructions

Big Idea: Stored-Program Concept

- Computers built on 2 key principles:
  - Instructions are represented as bit patterns - can think of these as numbers.
  - Therefore, entire programs can be stored in memory to be read or written just like data.
- Simplifies SW/HW of computer systems:
  - Memory technology for data also used for programs

Consequence #1: Everything Addressed

- Since all instructions and data are stored in memory, everything has a memory address: instructions, data words
  - both branches and jumps use these
- C pointers are just memory addresses: they can point to anything in memory
  - Unconstrained use of addresses can lead to nasty bugs; up to you in C; limits in Java
- One register keeps address of instruction being executed: **"Program Counter" (PC)**
  - Basically a pointer to memory: Intel calls it Instruction Address Pointer, a better name

Consequence #2: Binary Compatibility

- Programs are distributed in binary form
    - Programs bound to specific instruction set
    - Different version for Macintoshes and PCs
- New machines want to run old programs ("binaries") as well as programs compiled to new instructions
- Leads to "backward compatible" instruction set evolving over time
- Selection of Intel 8086 in 1981 for 1st IBM PC is major reason latest PCs still use 80x86 instruction set (Pentium 4); could still run program from 1981 PC today

Instructions as Numbers (1/2)

- Currently all data we work with is in words (32-bit blocks):
  - Each register is a word.
  - `lw` and `sw` both access memory one word at a time.
- So how do we represent instructions?
  - Remember: Computer only understands 1s and 0s, so "`add $t0,$0,$0`" is meaningless.
  - MIPS wants simplicity: since data is in words, make instructions be words too

Instructions as Numbers (2/2)

- One word is 32 bits, so divide instruction word into "fields".

- Each field tells processor something about instruction.

- We could define different fields for each instruction, but MIPS is based on simplicity, so define 3 basic types of instruction formats:

  - R-format
  - I-format
  - J-format

Instruction Formats

- I-format: used for instructions with immediates, `lw` and `sw` (since offset counts as an immediate), and branches (`beq` and `bne`),
  - (but not the shift instructions; later)
- J-format: used for `j` and `jal`
- R-format: used for all other instructions
- It will soon become clear why the instructions have been partitioned in this way.

R-Format Instructions (1/5)

- Define "fields" of the following number of bits each: 6 + 5 + 5 + 5 + 5 + 6 = 32

6	5	5	5	5	6

- For simplicity, each field has a name:

opcode	rs	rt	rd	shamt	funct

- Important: On these slides and in book, each field is viewed as a 5- or 6-bit unsigned integer, not as part of a 32-bit integer.

  - Consequence: 5-bit fields can represent any number 0-31, while 6-bit fields can represent any number 0-63.

R-Format Instructions (2/5)

- What do these field integer values tell us?
  - `opcode`: partially specifies what instruction it is
    - Note: This number is equal to 0 for all R-Format instructions.
  - `funct`: combined with `opcode`, this number exactly specifies the instruction
- Question: Why aren't `opcode` and `funct` a single 12-bit field?
  - We'll answer this later.

R-Format Instructions (3/5)

- More fields:
  - `rs` (**S**ource **R**egister): *generally* used to specify register containing first operand
  - `rt` (**T**arget **R**egister): *generally* used to specify register containing second operand (note that name is misleading)
  - `rd` (**D**estination **R**egister): *generally* used to specify register which will receive result of computation

R-Format Instructions (4/5)

- Notes about register fields:
  - Each register field is exactly 5 bits, which means that it can specify any unsigned integer in the range 0-31. Each of these fields specifies one of the 32 registers by number.
  - The word "generally" was used because there are exceptions that we'll see later. E.g.,
    - `mult` and `div` have nothing important in the `rd` field since the dest registers are `hi` and `lo`
    - `mfhi` and `mflo` have nothing important in the `rs` and `rt` fields since the source is determined by the instruction (see COD)

R-Format Instructions (5/5)

- Final field:
  - <u>shamt</u>: This field contains the amount a shift instruction will shift by. Shifting a 32-bit word by more than 31 is useless, so this field is only 5 bits (so it can represent the numbers 0-31).
  - This field is set to 0 in all but the shift instructions.
- For a detailed description of field usage for each instruction, see green insert in COD (You can bring with you to all exams)

R-Format Example (1/2)

- MIPS Instruction:

  add $8,$9,$10

  opcode = 0 (look up in table in book)

  funct = 32 (look up in table in book)

  rd = 8 (destination)

  rs = 9 (first *operand*)

  rt = 10 (second *operand*)

  shamt = 0 (not a shift)

R-Format Example (2/2)

- MIPS Instruction:

`add     $8,$9,$10`

Decimal number per field representation:

0	9	10	8	0	32

Binary number per field representation:

000000	01001	01010	01000	00000	100000

hex

hex representation: $012A \ 4020_{hex}$

decimal representation: $19,546,144_{ten}$

Called a <u>Machine Language Instruction</u>

Administrivia

- Remember to look at Appendix A (also on SPIM website), for MIPS assembly language details, including "assembly directives", etc.

- Other administrivia, TAs?

- How did you do in the faux exam?
  a) A+ = Aced it! (100%)
  b) A = Small errors, but mostly got it all right
  c) B = Some minor things to study, mostly ok
  d) C = "I've got some serious studying to do"
  e) Didn't take it

I-Format Instructions (1/4)

- What about instructions with immediates?
  - 5-bit field only represents numbers up to the value 31: immediates may be much larger than this
  - Ideally, MIPS would have only one instruction format (for simplicity): unfortunately, we need to compromise
- Define new instruction format that is partially consistent with R-format:
  - First notice that, if instruction has immediate, then it uses at most 2 registers.

I-Format Instructions (2/4)

- Define "fields" of the following number of bits each: 6 + 5 + 5 + 16 = 32 bits

6	5	5	16

  - Again, each field has a name:

opcode	rs	rt	immediate

  - Key Concept: Only one field is inconsistent with R-format. Most importantly, `opcode` is still in same location.

I-Format Instructions (3/4)

- What do these fields mean?
  - `opcode`: same as before except that, since there's no `funct` field, `opcode` uniquely specifies an instruction in I-format
  - This also answers question of why R-format has two 6-bit fields to identify instruction instead of a single 12-bit field: in order to be consistent as possible with other formats while leaving as much space as possible for immediate field.
  - `rs`: specifies a register operand (if there is one)
  - `rt`: specifies register which will receive result of computation (this is why it's called the *target* register "`rt`") or other operand for some instructions.

I-Format Instructions (4/4)

- The Immediate Field:

  - **addi, slti, sltiu**, the immediate is sign-extended to 32 bits. Thus, it's treated as a signed integer.

  - 16 bits ➔ can be used to represent immediate up to 2^{16} different values

  - This is large enough to handle the offset in a typical **lw** or **sw**, plus a vast majority of values that will be used in the **slti** instruction.

  - We'll see what to do when the number is too big in our next lecture…

I-Format Example (1/2)

- MIPS Instruction:

  addi $21,$22,-50

  opcode = 8 (look up in table in book)

  rs = 22 (register containing operand)

  rt = 21 (target register)

  immediate = -50 (by default, this is decimal)

I-Format Example (2/2)

- MIPS Instruction:

`addi     $21,$22,-50`

Decimal/field representation:

8	22	21	-50

Binary/field representation:

001000	10110	10101	1111111111001110

hexadecimal representation: $22D5\ FFCE_{hex}$
decimal representation: $584,449,998_{ten}$

Peer Instruction

Which instruction has same representation as 35_{ten}?

a) add $0, $0, $0

opcode	rs	rt	rd	shamt	funct

b) subu $s0,$s0,$s0

opcode	rs	rt	rd	shamt	funct

c) lw $0, 0($0)

opcode	rs	rt	offset		

d) addi $0, $0, 35

opcode	rs	rt	immediate		

e) subu $0, $0, $0

opcode	rs	rt	rd	shamt	funct

Registers numbers and names:
0: $0, .. 8: $t0, 9:$t1, ..15: $t7, 16: $s0, 17: $s1, .. 23: $s7

Opcodes and function fields (if necessary)

add: opcode = 0, funct = 32

subu: opcode = 0, funct = 35

addi: opcode = 8

lw: opcode = 35

Peer Instruction Answer

Which instruction has same representation as 35_{ten}?

a) add $0, $0, $0

0	0	0	0	0	32

b) subu $s0,$s0,$s0

0	16	16	16	0	35

c) lw $0, 0($0)

35	0	0		0

d) addi $0, $0, 35

8	0	0		35

e) subu $0, $0, $0

0	0	0	0	0	35

Registers numbers and names:
0: $0, .. 8: $t0, 9:$t1, ..15: $t7, 16: $s0, 17: $s1, .. 23: $s7

Opcodes and function fields (if necessary)

add: opcode = 0, funct = 32

subu: opcode = 0, funct = 35

**addi**: opcode = 8

lw: opcode = 35

In conclusion…

- Simplifying MIPS: Define instructions to be same size as data word (one word) so that they can use the same memory (compiler can use lw and sw).

- Computer actually stores programs as a series of these 32-bit numbers.

- MIPS Machine Language Instruction: 32 bits representing a single instruction

	opcode	rs	rt	rd	shamt	funct
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		

EXCLUSIVE!

ALTAIR 8800
The most powerful minicomputer project ever presented—can be built for under $400
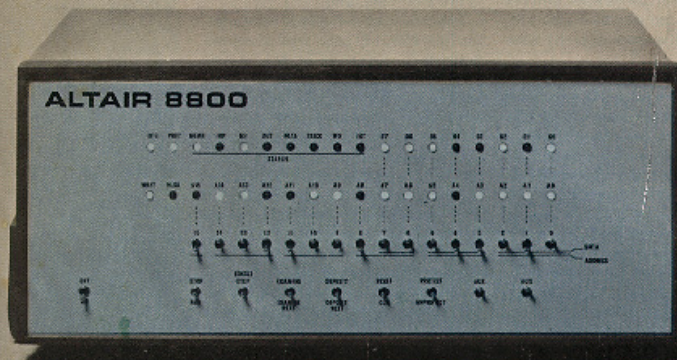
ALTAIR 8800

BY H. EDWARD ROBERTS AND WILLIAM YATES

THE era of the computer in every home—a favorite topic among science-fiction writers—has arrived! It's made possible by the POPULAR ELECTRONICS/MITS Altair 8800, a full-blown computer that can hold its own against sophisticated minicomputers now on the market. And it doesn't cost several thousand dollars. In fact, it's in a color TV-receiver's price class —under $400 for a complete kit.

The Altair 8800 is not a "demonstrator" or souped-up calculator. It is the most powerful computer ever presented as a construction project in any electronics magazine. In many ways, it represents a revolutionary development in electronic design and thinking.

The Altair 8800 is a parallel 8-bit word/16-bit address computer with an instruction cycle time of 2 μs. Its cen-

tral processing unit is a new LSI chip that is many times more powerful than previous IC processors. It can accommodate 256 inputs and 256 outputs, all directly addressable, and has 78 basic machine instructions (as compared with 40 in the usual minicomputer). This means that you can write an extensive and detailed program. The basic computer has 256 words of memory, but it can be economically expanded for 65,000 words. Thus, with full expansion, up to 65,000 subroutines can all be going at the same time.

The basic computer is a complete system. The program can be entered via switches located on the front panel, providing a LED readout in binary format. The very-low-cost terminal presented in POPULAR ELECTRONICS last month can also be used.

PROCESSOR DESCRIPTION

Processor: 8 bit parallel
Max. memory: 65,000 words (all directly addressable)
Instruction cycle time: 2 μs (min.)
Inputs and outputs: 256 (all directly addressable)
Number of basic machine instructions: 78 (181 with variants)
Add/subtract time: 2 μs
Number of subroutine levels: 65,000
Interrupt structure: 8 hardwire vectored levels plus software levels
Number of auxiliary registers: 8 plus stack pointer, program counter and accumulator
Memory type: semiconductor (dynamic or static RAM, ROM, PROM)
Memory access time: 850 ns static RAM; 420 or 150 ns dynamic Ram