

The MIPS Instruction Set Architecture

Computer Science 104
Lecture 5

Today's Lecture

Admin

- HW #1 is due
- HW #2 assigned

Outline

- Review
- Instruction categories
- Specific Instructions

Reading

Chapter 2, Appendix B

© Alvin R. Lebeck

CPS 104

2

Basic ISA Classes

Accumulator:

1 address	add A	$acc \leftarrow acc + mem[A]$
1+x address	addx A	$acc \leftarrow acc + mem[A+x]$

Stack:

0 address	add	$tos \leftarrow tos + next$ (JAVA VM)
-----------	-----	---------------------------------------

General Purpose Register:

2 address	add A B	$A \leftarrow A + B$
3 address	add A B C	$A \leftarrow B + C$

Load/Store:

3 address	add Ra Rb Rc	$Ra \leftarrow Rb + Rc$
	load Ra Rb	$Ra \leftarrow mem[Rb]$
	store Ra Rb	$mem[Rb] \leftarrow Ra$

© Alvin R. Lebeck

CPS 104

3

Accumulator

Instruction set: Accumulator is implicit operand

one explicit operand
add, sub, mult, div, ...
clear, store (st)

Example: $a*b - (a+c*b)$

Memory	
a	4
b	3
c	2
tmp	

9 instructions

© Alvin R. Lebeck

CPS 104

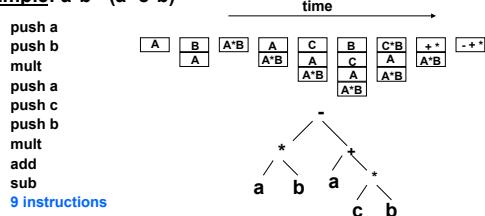
4

Stack Instruction Set Architecture

Instruction set:

add, sub, mult, div ... **Top of stack (TOS) and TOS+1 are implicit**
push A, pop A **TOS is implicit operand, one explicit operand**

Example: $a*b - (a+c*b)$



© Alvin R. Lebeck

CPS 104

5

2-address ISA

Instruction set: Two explicit operands, one implicit

add, sub, mult, div, ...
one source operand is also destination
add a,b a <- a + b

Example: $a*b - (a+c*b)$

Memory	
tmp1	3, ?
mult tmp1, c	6, ?
add tmp1, a	10, ?
add tmp2, b	10, 3
mult tmp2, a	10, 12
sub tmp2, tmp1	10, 2

6 instructions

© Alvin R. Lebeck

CPS 104

6

3-address ISA

- Instruction set: Three explicit operands, ZERO implicit

add, sub, mult, div, ...
add a,b,c a <- b + c

Example: $a*b - (a+c*b)$

tmp1, tmp2	a b c
mult tmp1, b, c 6, ?	4 3 2
add tmp1, tmp1, a 10, ?	
mult tmp2, a, b 10, 12	tmp1 tmp2
sub tmp2, tmp2, tmp1 10, 2	
4 instructions	

© Alvin R. Lebeck

CPS 104

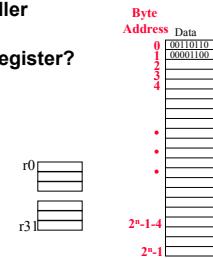
7

Adding Registers to an ISA

- A place to hold values that can be named within the instruction

- Like memory, but much smaller
➢ 32-128 locations

- How many bits to specify a register?



© Alvin R. Lebeck

CPS 104

8

3-address General Purpose Register ISA

- Instruction set: Three explicit operands, ZERO implicit

add, sub, mult, div, ...
add a,b,c a <- b + c

Example: $a*b - (a+c*b)$

r1, r2	a b c
mult r1, b, c 6, ?	4 3 2
add r1, r1, a 10, ?	
mult r2, a, b 10, 12	
sub r2, r2, r1 10, 2	
4 instructions	

© Alvin R. Lebeck

CPS 104

9

LOAD / STORE ISA

- Instruction set:

add, sub, mult, div, ... only on operands in registers
ld, st, to move data from and to memory, only way to access memory

Example: $a*b - (a+c*b)$ (assume in memory)

r1, r2, r3	a b c
ld r1, c 2, ?, ?	4 3 2
ld r2, b 2, 3, ?	
mult r1, r1, r2 6, 3, ?	
ld r3, a 6, 3, 4	
add r1, r1, r3 10, 3, 4	
mult r2, r2, r3 10, 12, 4	
sub r3, r2, r1 10, 12, 2	
7 instructions	

© Alvin R. Lebeck

CPS 104

10

Using Registers to Access Memory

- Registers can hold memory addresses

Given

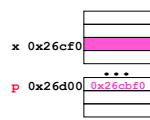
```
int x; int *p;
p = &x;
*p = *p + 8;
```

Instructions

```
ld r1, p // r1 <- mem[p]
ld r2, r1 // r2 <- mem[r1]
add r2, r2, 0x8 // increment x by 8
st r1, r2 // mem[r1] <- r2
```

- Many different ways to address operands

➢ not all Instruction sets include all modes



© Alvin R. Lebeck

CPS 104

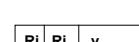
11

Addressing Modes

- Register direct Ri

- Immediate (literal) v

- Direct (absolute) M[v]



- Register indirect M[Ri]

- Base+Displacement M[Ri + v]

- Base+Index M[Ri + Rj]

- Scaled Index M[Ri + Rj*d + v]

- Autoincrement M[Ri+1]

- Autodecrement M[Ri - 1]

- Memory Indirect M[M[Ri]]

© Alvin R. Lebeck

CPS 104

12

Making Instructions Machine Readable

- So far, still too abstract
 - add r1, r2, r3
- Need to specify instructions in machine readable form
- Bunch of Bits
- Instructions are bits with well defined fields
 - Like a floating point number has different fields
- Instruction Format
 - establishes a mapping from "instruction" to binary values
 - which bit positions correspond to which parts of the instruction (operation, operands, etc.)

© Alvin R. Lebeck

CPS 104

13

Example: MIPS

Register-Register

31	26	25	21	20	16	15	11	10	6	5	0
Op	Rs1	Rs2	Rd						Opx		

Register-Immediate

31	26	25	21	20	16	15	0
Op	Rs1	Rd					immediate

Branch

31	26	25	21	20	16	15	0
Op	Rs1	Rs2/Opx					immediate

Jump / Call

31	26	25	0
Op			target

© Alvin R. Lebeck

CPS 104

14

A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 64-bit GPR (R0 contains zero)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
 - no indirection

see: SPARC, MIPS, MC88100, AMD2900, i960, i860
PA-RISC, POWERPC, DEC Alpha, Clipper,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

© Alvin R. Lebeck

CPS 104

15

Stored Program Computer

- **Instructions:** a fixed set of built-in operations
- Instructions and data are stored in the (same) computer memory
- Fetch-Execute Cycle

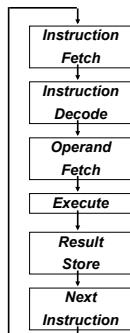

```
while (!done)
    fetch instruction
    execute instruction
```
- This is done by the hardware for speed
- This is what the SPIM Simulator does

© Alvin R. Lebeck

CPS 104

16

What Must be Specified?



- Instruction Format
 - how do we tell what operation to perform?
- Location of operands and result
 - where other than memory?
 - how many explicit operands?
 - how are memory operands located?
 - which can or cannot be in memory?
- Data type and Size
- Operations
 - what are supported
- Successor instruction
 - jumps, conditions, branches
- **fetch-decode-execute is implicit!**

© Alvin R. Lebeck

CPS 104

17

MIPS ISA Categories

- Arithmetic
 - add, sub, mul, etc
- Logical
 - and, or, shift
- Data Transfer
 - load, store
 - MIPS is LOAD/STORE architecture
- Conditional Branch
 - implement if, for, while... statements
- Unconditional Jump
 - support method invocation (function call, procedure calls)

© Alvin R. Lebeck

CPS 104

18

MIPS Instruction set Architecture

- 3-Address Load/Store Architecture.
- Register and Immediate addressing modes for operations.
- Immediate and Displacement addressing for Loads and Stores.
- Examples (Assembly Language):

```

add    $1, $2, $3      #      $1 = $2 + $3
addi   $1, $1, 4#      $1 = $1 + 4

lw     $1, 100($2)    # $1 = Memory[$2 + 100]
sw     $1, 100($2)    Memory[$2 + 100] = $1

lui    $1, 100        #      $1 = 100 X 216
addi   $1, $3, 100    #      $1 = $3 + 100

```

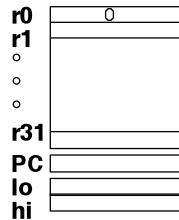
© Alvin R. Lebeck

CPS 104

19

MIPS Integer Registers

- Registers: fast memory, Integral part of the CPU.
- Programmable storage 2^{32} bytes
- 31 x 32-bit GPRs ($R0 = 0$)
- 32 x 32-bit FP regs (paired DP)
- 32-bit HI, LO, PC



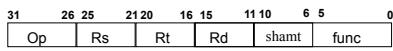
© Alvin R. Lebeck

CPS 104

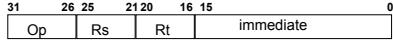
20

MIPS Instruction Formats

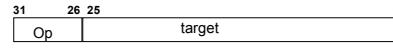
R-type: Register-Register



I-type: Register-Immediate



J-type: Jump / Call



Terminology

Op = opcode

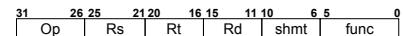
Rs, Rt, Rd = register specifier

© Alvin R. Lebeck

CPS 104

21

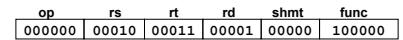
R Type: <OP> rd, rs, rt



op	a 6-bit operation code.
rs	a 5-bit source register.
rt	a 5-bit target (source) register.
rd	a 5-bit destination register.
shamt	a 5-bit shift amount.
func	a 6-bit function field.

Operand Addressing: Register direct

Example: ADD \$1, \$2, \$3 # \$1 = \$2 + \$3

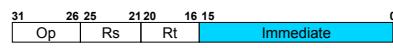


© Alvin R. Lebeck

CPS 104

22

I-Type <op> rt, rs, immediate



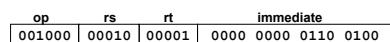
Immediate: 16 bit value

Operand Addressing:

Register Direct and Immediate

Add Immediate Example

addi \$1, \$2, 100 # \$1 = \$2 + 100

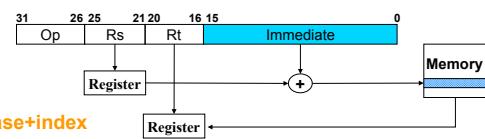


© Alvin R. Lebeck

CPS 104

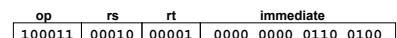
23

I-Type <op> rt, rs, immediate



Load Word Example

lw \$1, 100(\$2) # \$1 = Mem[\$2+100]

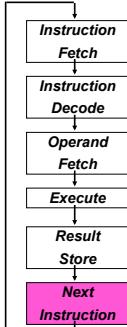


© Alvin R. Lebeck

CPS 104

24

Successor Instruction



```

main()
{
    int x,y,same;      // $0 == 0 always
    x = 43;           // addi $1, $0, 43
    y = 2;            // addi $2, $0, 2
    same = 0;          // addi $3, $0, 0
    if (x == y)
        same = 1;    // execute only if x == y
                      // addi $3, $0, 1
}
  
```

© Alvin R. Lebeck

CPS 104

25

The Program Counter (PC)

- Special register (PC) that points to instructions
- Contains memory address (like a pointer)
- Instruction fetch is
 - $\text{inst} = \text{mem}[pc]$
- To fetch next sequential instruction $\text{PC} = \text{PC} + ?$
 - Size of instruction?

© Alvin R. Lebeck

CPS 104

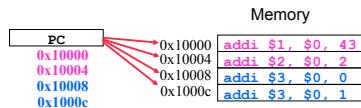
26

The Program Counter

```

x = 43;      // addi $1, $0, 43
y = 2; // addi $2, $0, 2
same = 0; // addi $3, $0, 0
if (x == y)
    same = 1; // addi $3, $0, 1 execute if x == y
  
```

PC is always automatically incremented to next instruction



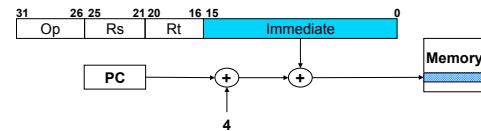
Clearly, this is not correct
We cannot always execute both 0x10008 and 0x1000c

© Alvin R. Lebeck

CPS 104

27

I-Type <op> rt, rs, immediate



- PC relative addressing

Branch Not Equal Example

bne \$1, \$2, 100 # If (\$1!= \$2) goto [PC+4+100]

- +4 because by default we increment for sequential
- more detailed discussion later in semester

op	rs	rt	immediate
000101	00001	00010	0000 0000 0110 0100

© Alvin R. Lebeck

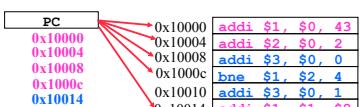
CPS 104

28

The Program Counter

```

x = 43;      // addi $1, $0, 43
y = 2; // addi $2, $0, 2
same = 0; // addi $3, $0, 0
if (x == y)
    same = 1; // addi $3, $0, 1 execute if x == y
    x = x + y; // addi $1, $1, $2
  
```



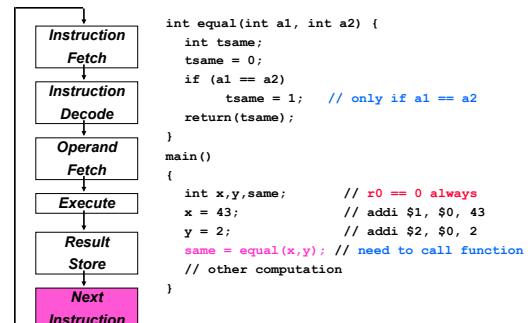
Understand branches

© Alvin R. Lebeck

CPS 104

29

Successor Instruction



© Alvin R. Lebeck

CPS 104

30

The Program Counter

- Branches are limited to 16 bit immediate
- Big programs?

```
x = 43; // addi $1, $0, 43
y = 2; // addi $2, $0, 2
same = equal(x,y);
```

0x10000	addi \$1, \$0, 43
0x10004	addi \$2, \$0, 2
0x10008	"go execute equal"

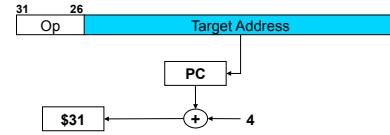
0x30408	addi \$3, \$0, 0
0x3040c	bne \$1, \$2, 4
0x30410	addi \$3, \$0, 1
	"return \$3"

© Alvin R. Lebeck

CPS 104

31

J-Type: <op> target



Jump and Link Example

JAL 1000 # PC<- 1000, \$31<-PC+4

\$31 set as side effect, used for returning, implicit operand

op	Target
0000011	00 0000 0000 0000 0011 1110 1000

© Alvin R. Lebeck

CPS 104

32

R Type: <OP> rd, rs, rt

31	26 25	21 20	16 15	11 10	6 5	0
Op	Rs	Rt	Rd	shamt	func	

Jump Register Example
jr \$31 # PC <- \$31

op	rs	rt	rd	shamt	func
000000	00010	10000	00001	00000	001000

© Alvin R. Lebeck

CPS 104

33

Instructions for Procedure Call and Return

int equal(int a1, int a2) {	0x10000 addi \$1, \$0, 43
int tsame;	0x10004 addi \$2, \$0, 2
tsame = 0;	0x10008 jal 0x30408
if (a1 == a2)	0x1000c ??
tsame = 1;	0x30408 addi \$3, \$0, 0
return(tsame);	0x3040c bne \$1, \$2, 4
}	0x30410 addi \$3, \$0, 1
main()	0x30414 jr \$31
{	PC \$31
int x,y,same;	0x10000 ??
x = 43;	0x10004 ??
y = 2;	0x10008 ??
same = equal(x,y);	0x30408 0x1000c
// other computation	0x3040c 0x1000c
}	0x30410 0x1000c
	0x30414 0x1000c
	0x1000c 0x1000c

© Alvin R. Lebeck

CPS 104

34

MIPS Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands
subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands
add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant
add unsigned	addu \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands
subtract unsigned	subu \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands
add imm. unsigned.	addiu \$1,\$2,100	\$1 = \$2 + 100	+ constant
multiply	mult \$2,\$3	Hi, Lo = \$2 x \$3	64-bit signed product
multiply unsigned	multu \$2,\$3	Hi, Lo = \$2 x \$3	64-bit unsigned product
divide	div \$2,\$3	Lo = \$2 ÷ \$3, Lo = quotient, Hi = \$2 mod \$3 Hi = remainder	
divide unsigned	divu \$2,\$3	Lo = \$2 ÷ \$3, Unsigned quotient Hi = \$2 mod \$3 Unsigned remainder	
Move from Hi	mfhi \$1	\$1 = Hi	Used to get copy of Hi
Move from Lo	mflo \$1	\$1 = Lo	Used to get copy of Lo

Which add for address arithmetic? Which for integers?

© Alvin R. Lebeck

CPS 104

35

MIPS Logical Instructions

Instruction	Example	Meaning	Comment
and	and \$1,\$2,\$3	\$1 = \$2 & \$3	Bitwise AND
or	or \$1,\$2,\$3	\$1 = \$2 \$3	Bitwise OR
xor	xor \$1,\$2,\$3	\$1 = \$2 ⊕ \$3	Bitwise XOR
nor	nor \$1,\$2,\$3	\$1 = ~(\$2 \$3)	Bitwise NOR
and immediate	andi \$1,\$2,10	\$1 = \$2 & 10	Bitwise AND reg, const
or immediate	ori \$1,\$2,10	\$1 = \$2 10	Bitwise OR reg, const
xor immediate	xori \$1,\$2,10	\$1 = -\$2 & -10	Bitwise XOR reg, const
shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
shift right logical	srl \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant
shift right arithm.	sra \$1,\$2,10	\$1 = \$2 >> 10	Shift right (sign extend)
shift left logical	sllv \$1,\$2,\$3	\$1 = \$2 << \$3	Shift left by var
shift right logical	srlv \$1,\$2,\$3	\$1 = \$2 >> \$3	Shift right by var
shift right arithm.	srav \$1,\$2,\$3	\$1 = \$2 >> \$3	Shift right arith. by var

© Alvin R. Lebeck

CPS 104

36

MIPS Data Transfer Instructions

Instruction	Comment
SW R3, 500(R4)	Store word
SH R3, 502(R2)	Store half
SB R2, 41(R3)	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load halfword
LHU R1, 40(R3)	Load halfword unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)

Why do we need LUI?



© Alvin R. Lebeck

CPS 104

37

MIPS Compare and Branch

Compare and Branch

```
beq rs, rt, offset    if R[rs] == R[rt] then PC-relative branch
bne rs, rt, offset    <>
```

Compare to zero and Branch

```
bnez rs, offset      if R[rs] != 0 then PC-relative branch
bgtz rs, offset      >
bltz rs, offset      <
bgez rs, offset      >=
bltzal rs, offset    if R[rs] < 0 then branch and link (into R 31)
bgeal rs, offset      >=
```

- Remaining set of compare and branch take two instructions

- Almost all comparisons are against zero!

© Alvin R. Lebeck

CPS 104

38

MIPS jump, branch, compare instructions

Instruction	Example	Meaning
branch on equal	<code>beq \$1,\$2,100</code>	if (\$1 == \$2) go to PC+4+100 Equal test; PC relative branch
branch on not eq.	<code>bne \$1,\$2,100</code>	if (\$1!= \$2) go to PC+4+100 Not equal test; PC relative
set on less than	<code>slt \$1,\$2,\$3</code>	if (\$2 < \$3) \$1=1; else \$1=0 Compare less than; 2's comp.
set less than imm.	<code>slti \$1,\$2,100</code>	if (\$2 < 100) \$1=1; else \$1=0 Compare < constant; 2's comp.
set less than uns.	<code>sltu \$1,\$2,\$3</code>	if (\$2 < \$3) \$1=1; else \$1=0 Compare less than; natural numbers
set l. t. imm. uns.	<code>sltu \$1,\$2,100</code>	if (\$2 < 100) \$1=1; else \$1=0 Compare < constant; natural numbers
jump	<code>j 10000</code>	go to 10000 Jump to target address
jump register	<code>jr \$31</code>	go to \$31 For switch, procedure return
jump and link	<code>jal 10000</code>	\$31 = PC + 4; go to 10000 For procedure call

© Alvin R. Lebeck

CPS 104

39

Signed v.s. Unsigned Comparison

R1= 0...00 0000 0000 0000 0001

R2= 0...00 0000 0000 0000 0010

R3= 1...11 1111 1111 1111 1111

- After executing these instructions:

```
slt r4,r2,r1
slt r5,r3,r1
sltu r6,r2,r1
sltu r7,r3,r1
```

- What are values of registers r4 - r7? Why?

r4 = ; r5 = ; r6 = ; r7 = ;

© Alvin R. Lebeck

CPS 104

40

Summary

- MIPS has 5 categories of instructions
 - Arithmetic, Logical, Data Transfer, Conditional Branch, Unconditional Jump
- 3 Instruction Formats

Next Time

- Assembly Programming
- Reading
 - Ch. 2, Appendix B
 - HW #2

© Alvin R. Lebeck

CPS 104

41