CPS104 Computer Organization

Lecture 6

Instruction Set Architecture
MIPS

September 10 , 2008

Gershon Kedem

©GK Fall 2008

Administratrivia

- Programming Puzzle-1 is on the Web, see: `kedem.cs.duke.edu/cps104/ppz/PPZ1.html`
- Due: September Today (11:59 pm).

- Programming Puzzle-2 is on the Web, see: `kedem.cs.duke.edu/cps104/ppz/PPZ2.html`
- Due: September 17 (11:59 pm).
- Submit the `*.java` file to `cps104/PPZ2`
- Use the Eclipse system to submit the file.
- Help is on-line.

©GK Fall 2008

Review: Mask and Shift example: Extracting Parts of Floating Point Number

```
// x is 32-bit word holding a float

#define EXP_BITS 8

#define MANTISSA_BITS 23

#define SIGN_MASK 0x80000000

#define EXP_MASK  0x7f800000

#define MANTISSA_MASK  0x007fffff

class myfloat {

public:

  int sign;

  unsigned int exp;

  unsigned int mantissa;

};

myfloat num;

 num->sign = (x & SIGN_MASK) >> (EXP_BITS + MANTISSA_BITS);

 num->exp  = (x & EXP_MASK) >> MANTISSA_BITS;

 num->mantissa = x & MANTISSA_MASK;
```

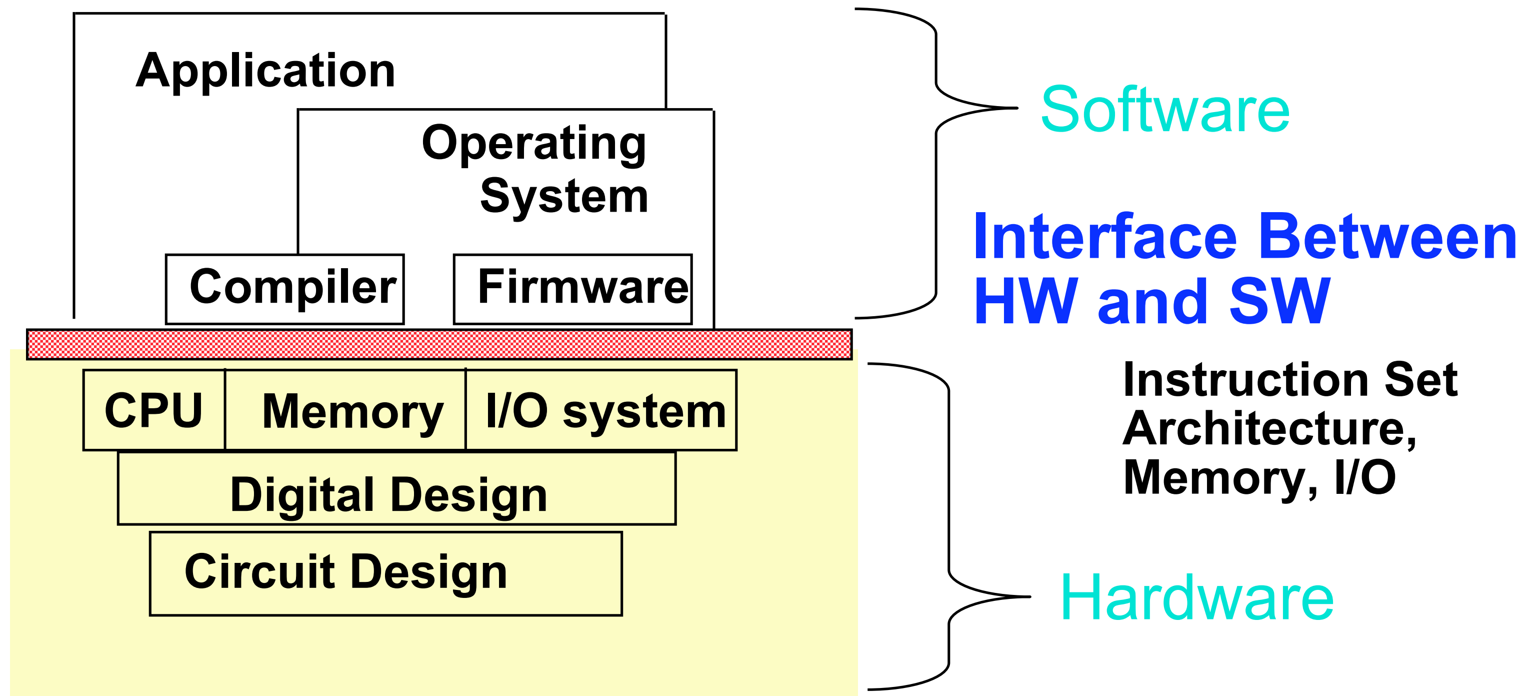

©GK Fall 2008

Summary

- Computer memory is a linear array of bytes
- Pointer is memory location that contains address of another memory location
- Bitwise operations
- We'll visit these topics again throughout semester.

©GK Fall 2008

Instruction Set Architecture



©GK Fall 2008

Levels of Representation

High Level Language Program

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

Compiler

Assembly Language Program

lw $15, 0($2)
lw $16, 4($2)
sw$16, 0($2)
sw$15, 4($2)

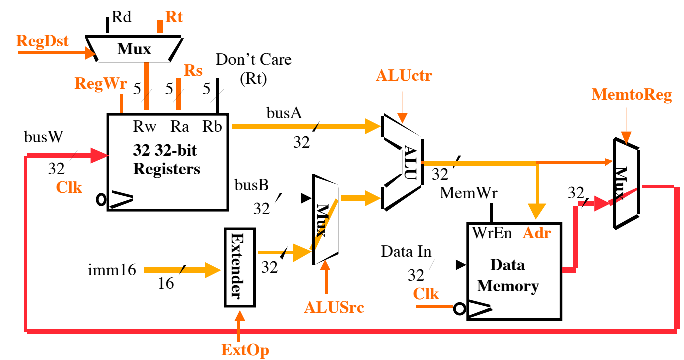
Assembler

Machine Language Program

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

Machine Interpretation

Control Signal Specification

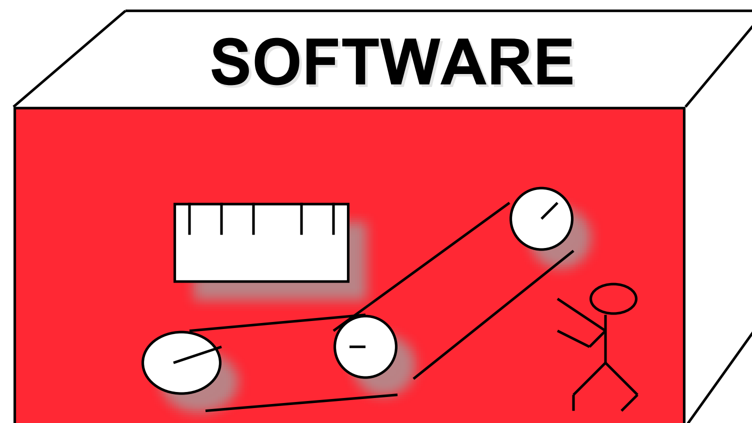


©GK Fall 2008

Computer Architecture?

. . . the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964



©GK Fall 2008

Requirements for ISA

```
#include <stdio.h>

main()
{
  int a[100];
  int k;

  for (k = 0; k < 100; k++)
    {
      a[k] = k*7+5;
    }

  printf("entry 3 = %d\n", a[3]);
}
```

What primitive operations do we need?
(i.e., What should be implemented in hardware?)

©GK Fall 2008

Design Space of ISA

Five Primary Dimensions

• Operations	add, sub, mul, . . .
	How is it specified?
• Number of explicit operands	(0, 1, 2, 3)
• Operand Storage	Where besides memory?
• Memory Address	How is memory location specified?
• Type & Size of Operands	byte, int, float, vector, . . .
	How is it specified?

Other Aspects

• Successor instruction	How is it specified?
• Conditions	How are they determined?
• Encoding	Fixed or variable? Wide?
• Parallelism	

©GK Fall 2008

Basic ISA Classes

Accumulator:

1 address	add A	acc ← acc + mem[A]
1+x address	addx A	acc ← acc + mem[A + x]

Stack:

0 address	add	tos ← tos + next (JAVA VM)

General Purpose Register:

2 address	add A B	A ← A + B
3 address	add A B C	A ← B + C

Load/Store:

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

©GK Fall 2008

Accumulator

- Instruction set: Accumulator is implicit operand

  one explicit operand

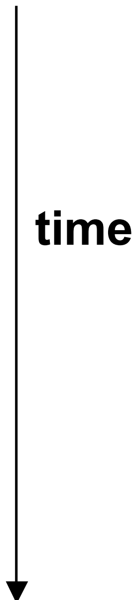
  add, sub, mult, div, . . .

  clear

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

clear	0
add c	2
mult b	6
add a	10
st tmp	10
clear	0
add a	4
mult b	12
sub tmp	2

9 instructions



a	4
b	3
c	2
tmp	

©GK Fall 2008

Stack Machines

- **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)

push a

push b

mult

push a

push c

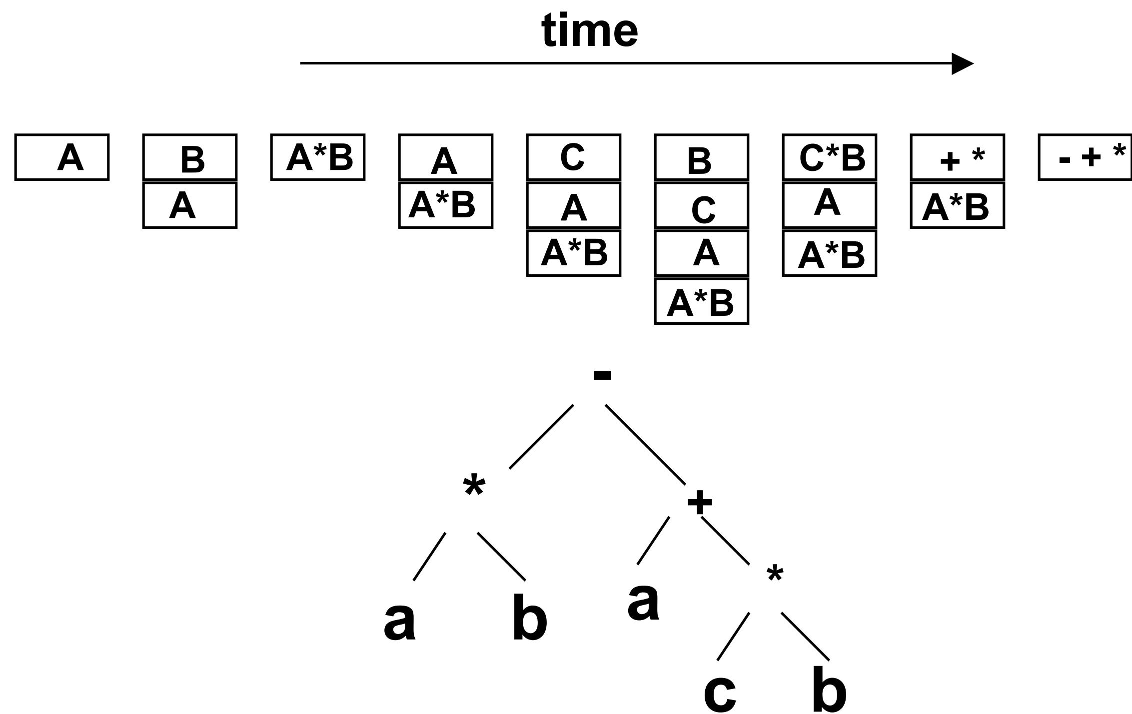
push b

mult

add

sub

9 instructions



©GK Fall 2008

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)

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

a	4
b	3
c	2
tmp1	
tmp2	

6 instructions

©GK Fall 2008

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

4 instructions

a	4
b	3
c	2
tmp1	
tmp2	

©GK Fall 2008

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?

r0

r31

Byte Address

Data

0 00110110

1 00001100

2

3

4

2^{32}-1-4

2^{32}-1

©GK Fall 2008

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)

Instruction	r1	r2
mult r1, b, c	6	?
add r1, r1, a	10	?
mult r2, a, b	10	12
sub r2, r2, r1	10	2

a	4
b	3
c	2

4 instructions

©GK Fall 2008

LOAD / STORE ISA

- **Instruction set:**

  add, sub, mult, div, … only on operands in registers

  ld, st, to move data from and to memory,
  The only way to access memory

<u>Example</u>: a*b - (a+c*b)

	<u>r1</u>, <u>r2</u>, <u>r3</u>
ld r1, c	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

a	4
b	3
c	2

7 instructions

©GK Fall 2008

Using Registers to Access Memory

- Registers can hold memory addresses

Given

```
int x; int *p;

p = &x;

*p = *p + 8;
```

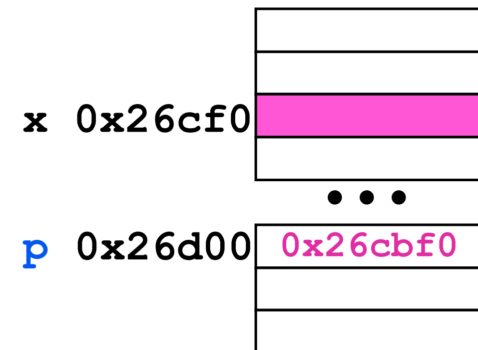
Instructions

```
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

©GK Fall 2008

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.)

©GK Fall 2008

A "Typical" RISC

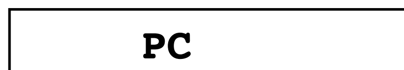
- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero) (could be 64-bit GPR)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store:
  base + displacement
    - no indirection

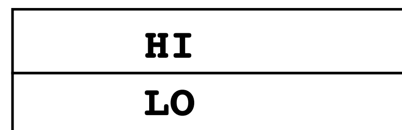
see: SPARC, MIPS, MC88100, AMD2900, i960, i860
PARISC, PowerPC, DEC Alpha, Clipper,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

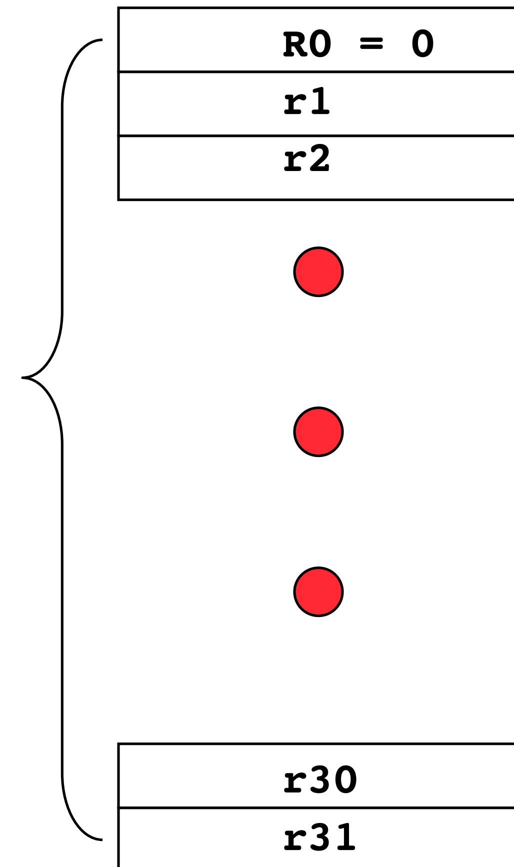
©GK Fall 2008

Example: MIPS 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 for DP)
- 32-bit HI, LO, PC



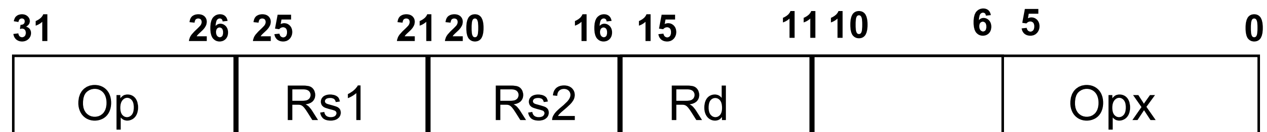




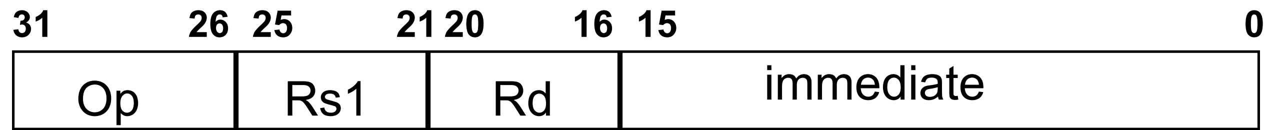
©GK Fall 2008

Example: MIPS Instructions' Formats

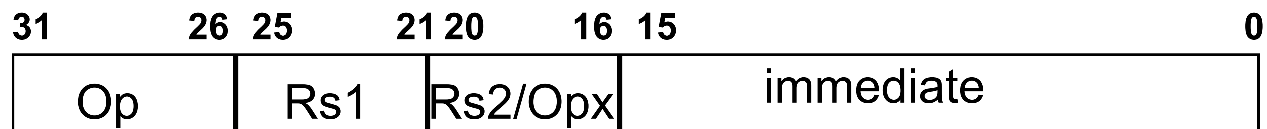
Register-Register



Register-Immediate



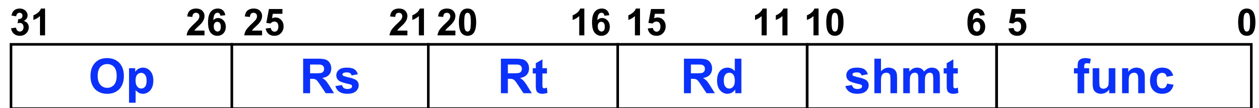
Branch



Jump / Call

©GK Fall 2008

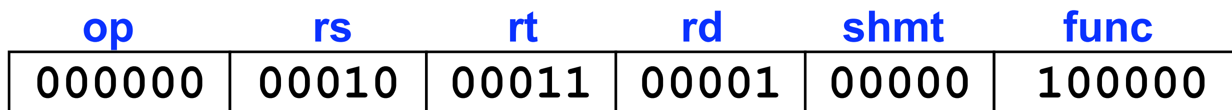
R Type: <OP> rd, rs, rt



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.
shmt	a 5-bit shift amount.
func	a 6-bit function field.

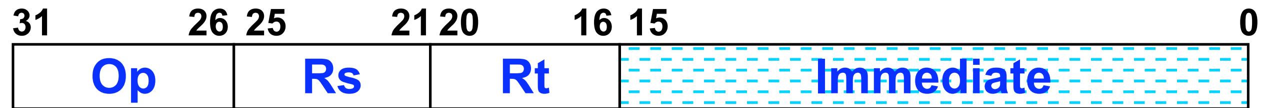
Operand Addressing: Register direct

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



©GK Fall 2008

I-Type <op> rt, rs, immediate

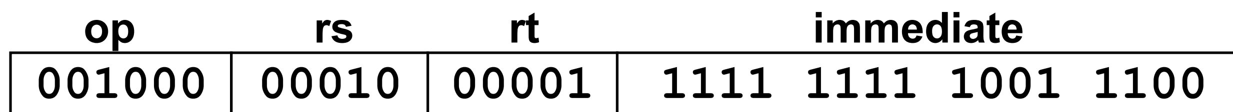


Immediate: 16 bit value

Operand Addressing: Register Direct and Immediate

Add Immediate Example

addi $1, $2, -100 # $1 = $2 + (-100)

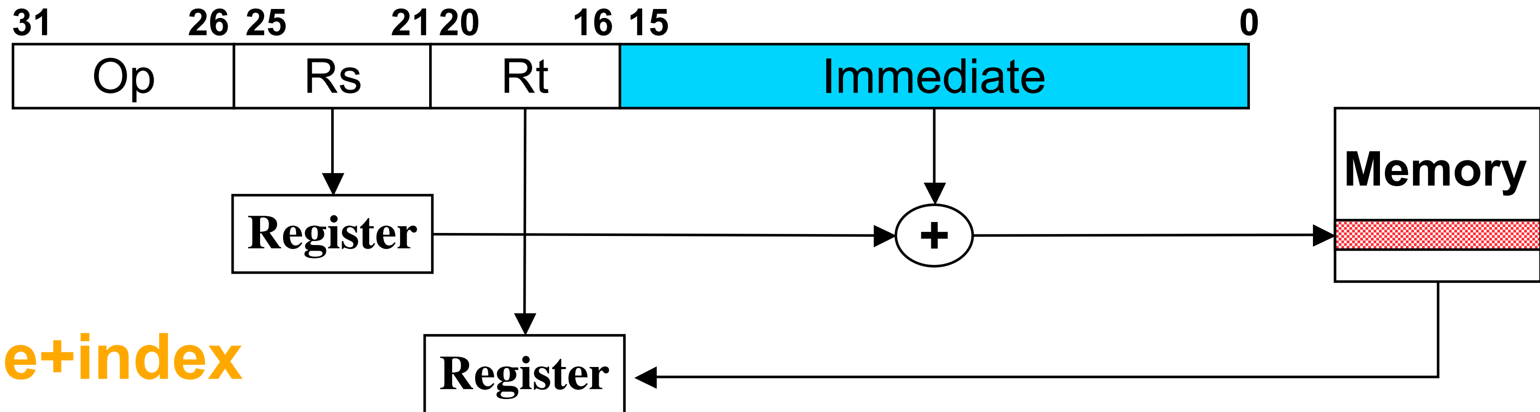


op	rs	rt	immediate
001000	00010	00001	1111 1111 1001 1100

Rt becomes the destination register!

©GK Fall 2008

I-Type <op> rt, rs, immediate



Base+index

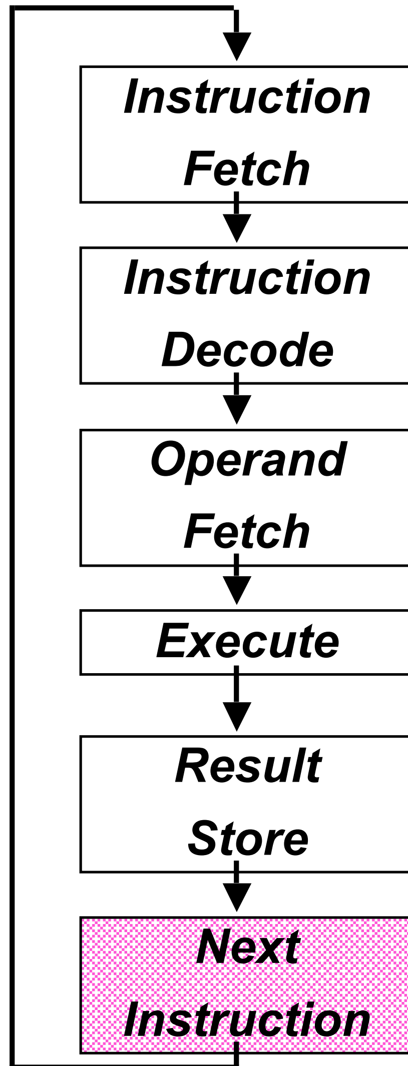
Load Word Example

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

op	rs	rt	immediate
010011	00010	00001	0000 0000 0110 0100

©GK Fall 2008

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
}
```


©GK Fall 2008

The Program Counter

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

©GK Fall 2008

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
```

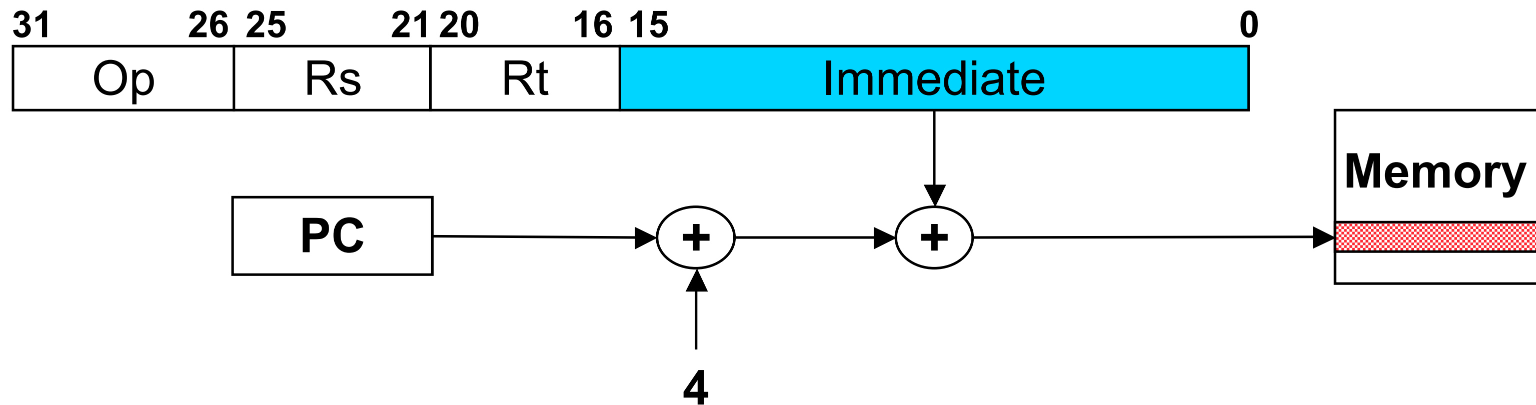
PC
0x10000
0x10004
0x10008
0x1000c

Address	Instruction
0x10000	addi $1, $0, 43
0x10004	addi $2, $0, 2
0x10008	addi $3, $0, 0
0x1000c	addi $3, $0, 1

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

©GK Fall 2008

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

©GK Fall 2008

The Program Counter

```
x = 43;        // addi $1, $0, 43

y = 2;         // addi $2, $0, 2

if (x == y)

  same = 1;  // addi $3, $0, $1 execute if x == y
```

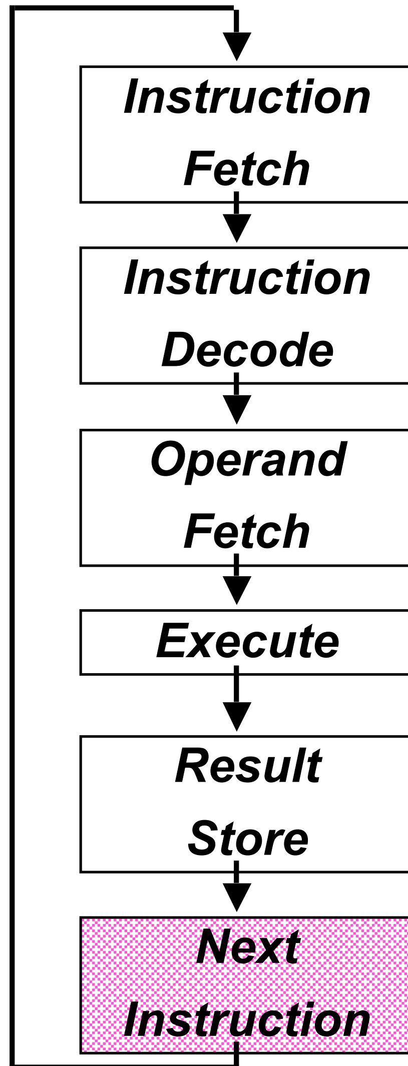
PC
0x10000
0x10004
0x10008
0x1000c
0x10010

Address	Instruction
0x10000	addi $1, $0, 43
0x10004	addi $2, $0, 2
0x10008	addi $3, $0, 0
0x1000c	bne $1, $2, 4
0x10010	addi $3, $0, 1

Understand branches

©GK Fall 2008

Successor Instruction




```
int equal(int a1, int a2) {
   int tsame;
   tsame = 0;
   if (a1 == a2)
           tsame = 1;   // only if a1 == a2
   return(tsame);
}
main()
{
   int x,y,same;      // r0 == 0 always
   x = 43;            // addi $1, $0, 43
   y = 2;             // addi $2, $0, 2
   same = equal(x,y); // need to call function
   // other computation
}
```


©GK Fall 2008

The Program Counter

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

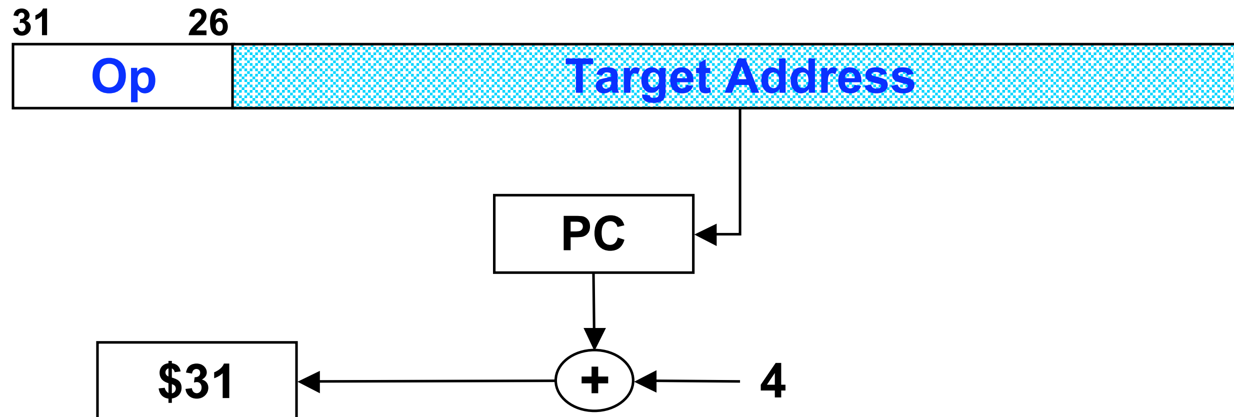
```
x = 43;  // addi $1, $0, 43
y = 2;   // addi $2, $0, 2
same = equal(x,y);
```

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

Address	Instruction
0x30408	addi $3, $0, 0
0x3040c	beq $1, $2, 8
0x30410	addi $3, $0, 1
	"return $3"

©GK Fall 2008

J-Type: <op> target



Jump and Link Example

jal 0x0fab8 # PC<- 0x0fab8, $31<-PC+4

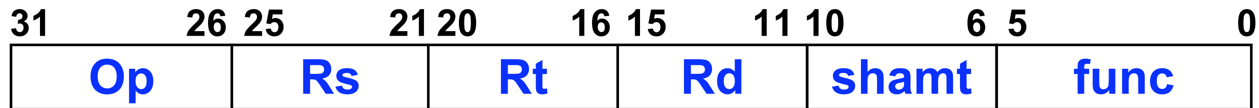
$31 set as side effect, used for returning, implicit operand

op	Target
000011	00 0000 0000 0011 1110 1010 1110

Please note, The address is a WORD ADDRESS!

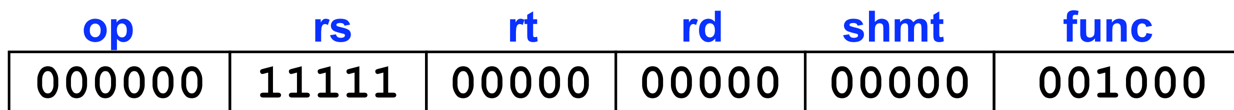
©GK Fall 2008

R Type: <OP> rd, rs, rt



Jump Register Example

jr $31 # PC <- $31



©GK Fall 2008