

CS301 Computer Architecture

DR GAYATHRI ANANTHANARAYANAN

gayathri@iitdh.ac.in

Materials in these slides are borrowed from textbooks and existing Architecture courses

SimpleRisc

- * Simple RISC ISA
- * Contains only 21 instructions
- * We will design an assembly language for SimpleRisc
- * Design a simple binary encoding,
- * and then implement it ...



Survey of Instruction Sets

ISA	Type	Year	Vendor	Bits	Endianness	Registers
VAX	CISC	1977	DEC	32	little	16
SPARC	RISC	1986	Sun	32	big	32
	RISC	1993	Sun	64	bi	32
PowerPC	RISC	1992	Apple,IBM,Motorola	32	bi	32
	RISC	2002	Apple,IBM	64	bi	32
PA-RISC	RISC	1986	HP	32	big	32
	RISC	1996	HP	64	big	32
m68000	CISC	1979	Motorola	16	big	16
	CISC	1979	Motorola	32	big	16
MIPS	RISC	1981	MIPS	32	bi	32
	RISC	1999	MIPS	64	bi	32
Alpha	RISC	1992	DEC	64	bi	32
x86	CISC	1978	Intel,AMD	16	little	8
	CISC	1985	Intel,AMD	32	little	8
	CISC	2003	Intel,AMD	64	little	16
ARM	RISC	1985	ARM	32	bi(little default)	16
	RISC	2011	ARM	64	bi(little default)	31

Registers

- * SimpleRisc has 16 registers
 - * Numbered : r0 ... r15
 - * r14 is also referred to as the stack pointer (sp)
 - * r15 is also referred to as the return address register (ra)
- * View of Memory
 - * Von Neumann model
 - * One large array of bytes
- * Special flags register → contains the result of the last comparison
 - * flags.E = 1 (equality), flags.GT = 1 (greater than)

mov instruction

mov r1,r2	$r1 \leftarrow r2$
mov r1,3	$r1 \leftarrow 3$

- * **Transfer** the contents of one **register** to another
- * Or, transfer the contents of an **immediate** to a register
- * The value of the **immediate** is embedded in the instruction
 - * SimpleRisc has 16 bit immediates
 - * Range -2^{15} to $2^{15} - 1$

Arithmetic/Logical Instructions

- * SimpleRisc has 6 arithmetic instructions
 - * add, sub, mul, div, mod, cmp

Example	Explanation
add r1, r2, r3	$r1 \leftarrow r2 + r3$
add r1, r2, 10	$r1 \leftarrow r2 + 10$
sub r1, r2, r3	$r1 \leftarrow r2 - r3$
mul r1, r2, r3	$r1 \leftarrow r2 \times r3$
div r1, r2, r3	$r1 \leftarrow r2 / r3$ (quotient)
mod r1, r2, r3	$r1 \leftarrow r2 \bmod r3$ (remainder)
cmp r1, r2	set flags

Examples of Arithmetic Instructions

- * Convert the following code to assembly

```
a = 3  
b = 5  
c = a + b  
d = c - 5
```

- * Assign the variables to registers

- * $a \leftarrow r0, b \leftarrow r1, c \leftarrow r2, d \leftarrow r3$

```
mov r0, 3  
mov r1, 5  
add r2, r0, r1  
sub r3, r2, 5
```

Examples - II

- * Convert the following code to assembly

```
a = 3  
b = 5  
c = a * b  
d = c mod 5
```

- * Assign the variables to registers

- * $a \leftarrow r0, b \leftarrow r1, c \leftarrow r2, d \leftarrow r3$

```
mov r0, 3  
mov r1, 5  
mul r2, r0, r1  
mod r3, r2, 5
```


Compare Instruction

- * Compare 3 and 5, and print the value of the flags

```
a = 3  
b = 5  
compare a and b
```

```
mov r0, 3  
mov r1, 5  
cmp r0, r1
```

- * flags.E = 0, flags.GT = 0

Compare Instruction

- * Compare 5 and 3, and print the value of the flags

```
a = 5  
b = 3  
compare a and b
```

```
mov r0, 5  
mov r1, 3  
cmp r0, r1
```

- * flags.E = 0, flags.GT = 1

Compare Instruction

- * Compare 5 and 5, and print the value of the flags

```
a = 5  
b = 5  
compare a and b
```

```
mov r0, 5  
mov r1, 5  
cmp r0, r1
```

- * flags.E = 1, flags.GT = 0

Example with Division

Write assembly code in SimpleRisc to compute: $31 / 29 - 50$, and save the result in r4.

Answer:

SimpleRisc

```
mov r1, 31
mov r2, 29
div r3, r1, r2
sub r4, r3, 50
```

Logical Instructions

and r1, r2, r3	$r1 \leftarrow r2 \& r3$
or r1, r2, r3	$r1 \leftarrow r2 r3$
not r1, r2	$r1 \leftarrow \sim r2$
& bitwise AND, bitwise OR, ~ logical complement	

- * The second argument can either be a register or an immediate

Compute $(a | b)$. Assume that a is stored in $r0$, and b is stored in $r1$. Store the result in $r2$.

Answer:

SimpleRisc
`or r2, r0, r1`

Shift Instructions

- * Logical shift left (lsl) (<< operator)
 - * $0010 \ll 2$ is equal to 1000
 - * $(\ll n)$ is the same as multiplying by 2^n
- * Arithmetic shift right (asr) (>> operator)
 - * $0010 \gg 1 = 0001$
 - * $1000 \gg 2 = 1110$
 - * same as dividing a signed number by 2^n

2's Complement Notation

$$F(u) = \begin{cases} u, & 0 \leq u \leq 2^{n-1} - 1 \\ 2^n - |u|, & -2^{n-1} \leq u < 0 \end{cases}$$

- * $F(u)$ is the index of a point on the **number circle**. It varies from 0 to $2^n - 1$

- * Examples

- * $4 \rightarrow 0100$

- * $-4 \rightarrow 1100$

- * $5 \rightarrow 0101$

- * $-3 \rightarrow 1101$

Properties of the 2's Complement Notation

- * Range of the number system :
 - * $-2^{(n-1)}$ to $2^{n-1} - 1$
- * There is a unique representation for 0
→ 000000
- * msb of $F(u)$ is equal to $\text{SgnBit}(u)$
 - * For a +ve number, $F(u) < 2^{(n-1)}$. MSB = 0
 - * For a -ve number, $F(u) \geq 2^{(n-1)}$. MSB = 1

Computing the 2's Complement

- * $2^n - u$

$$= 2^n - 1 - u + 1$$

$$= \sim u + 1$$

- * $\sim u$ (1's complement)

* 1's complement of 0100 2's complement of 0100

—	1111
	0100
—	
	1011

+	1011
	0001
—	
	1100

Shift Instructions

- * Logical shift left (lsl) (<< operator)
 - * $0010 \ll 2$ is equal to 1000
 - * $(\ll n)$ is the same as multiplying by 2^n
- * Arithmetic shift right (asr) (>> operator)
 - * $0010 \gg 1 = 0001$
 - * $1000 \gg 2 = 1110$
 - * same as dividing a signed number by 2^n

Shift Instructions - II

- * logical shift right (lsr) (>>> operator)
 - * $1000 \ggg 2 = 0010$
 - * same as dividing the unsigned representation by 2^n

Example	Explanation
lsl r3, r1, r2	$r3 \leftarrow r1 \ll r2$ (shift left)
lsl r3, r1, 4	$r3 \leftarrow r1 \ll 4$ (shift left)
lsr r3, r1, r2	$r3 \leftarrow r1 \ggg r2$ (shift right logical)
lsr r3, r1, 4	$r3 \leftarrow r1 \ggg 4$ (shift right logical)
asr r3, r1, r2	$r3 \leftarrow r1 \gg r2$ (arithmetic shift right)
asr r3, r1, 4	$r3 \leftarrow r1 \gg 4$ (arithmetic shift right)

Example with Shift Instructions

- * Compute $101 * 6$ with shift operators

```
mov r0, 101  
lsl r1, r0, 1  
lsl r2, r0, 2  
add r3, r1, r2
```

Example - II

- * Compute $102 * 7.5$ with shift operators

```
mov r0, 102  
lsl r1, r0, 3  
lsr r2, r0, 1  
sub r3, r1, r2
```

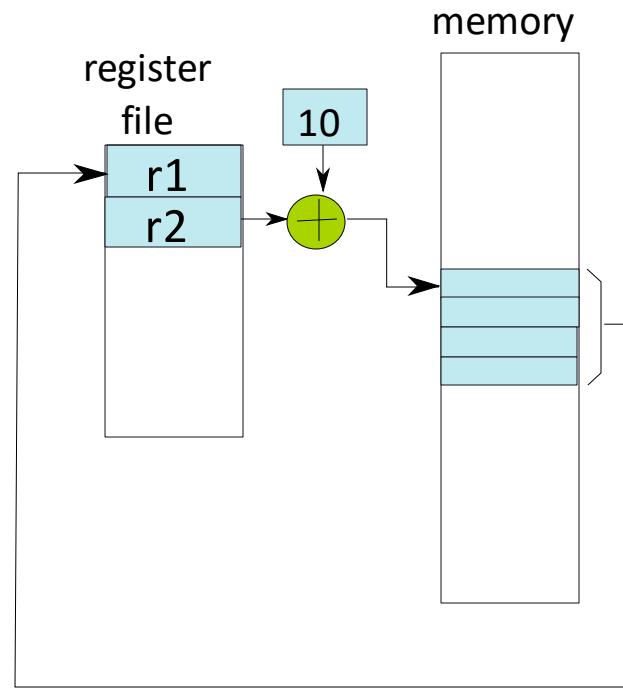
Load-store instructions

ld r1, 10[r2]	$r1 \leftarrow [r2 + 10]$
st r1, 10[r2]	$[r2 + 10] \leftarrow r1$

- * 2 address format, base-offset addressing
- * Fetch the contents of r2, add the offset (10), and then perform the memory access

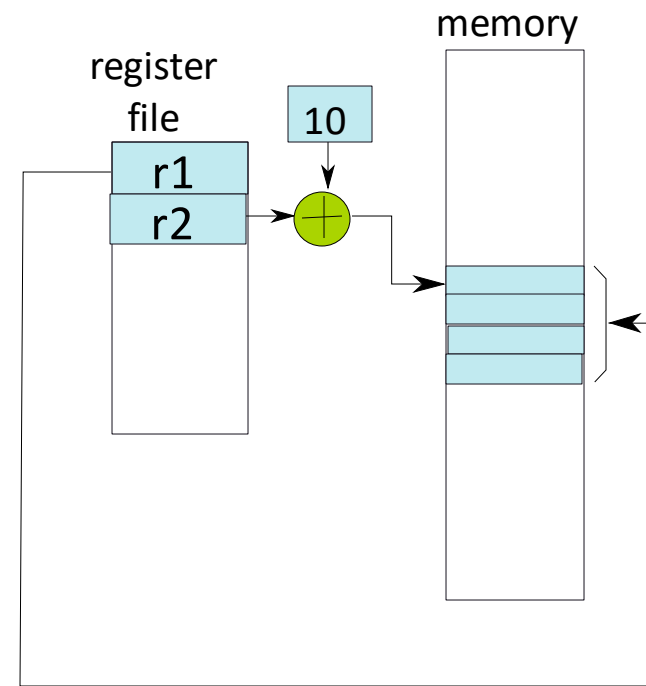
Load-Store

ld r1, 10[r2]



(a)

st r1, 10[r2]



(b)

Example – Load/Store

* Translate :

```
int arr[10];  
arr[3] = 5;  
arr[4] = 8;  
arr[5] = arr[4] + arr[3];
```

```
/* assume base of array saved in r0 */  
mov r1, 5  
st r1, 12[r0]  
mov r2, 8  
st r2, 16[r0]  
add r3, r1, r2  
st r3, 20[r0]
```


Branch Instructions

- * Unconditional branch instruction

b .foo	branch to .foo
--------	----------------

```
add r1, r2, r3
b .foo
...
...
.foo:
    add r3, r1, r4
```

Conditional Branch Instructions

beq .foo	branch to .foo if $flags.E = 1$
bgt .foo	branch to .foo if $flags.GT = 1$

- * The flags are only set by cmp instructions
- * beq (branch if equal)
 - * If $flags.E = 1$, jump to .foo
- * bgt (branch if greater than)
 - * If $flags.GT = 1$, jump to .foo

Examples

- * If $r1 > r2$, then save 4 in $r3$, else save 5 in $r3$

```
cmp r1, r2
bgt .gtlabel
mov r3, 5
...
...
.gtlable:
    mov r3, 4
```

Example - II

Answer: Compute the factorial of the variable num.


C

```
int prod = 1;
int idx;
for(idx = num; idx > 1; idx --) {
    prod = prod * idx
}
```

Let us now try to convert this program to SimpleRisc .

SimpleRisc

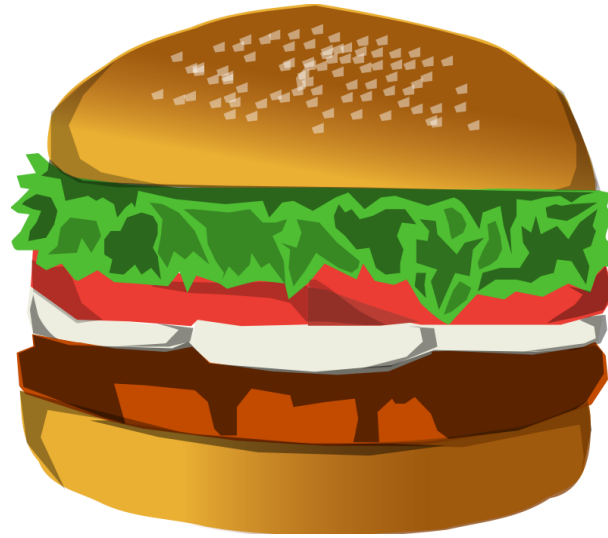
```
mov r1, 1          /* prod = 1 */
mov r2, r0         /* idx = num */
.loop:
    mul r1, r1, r2  /* prod = prod * idx */
    sub r2, r2, 1   /* idx = idx - 1 */
    cmp r2, 1       /* compare (idx, 1) */
    bgt .loop       /* if (idx > 1) goto .loop*/
```



You are given a number in register r1. You may assume that this number is greater than 3. Write a SimpleRISC program to find out if this number is a prime number or not. If it is, write 1 to r0. Else, write 0 to r0.

You are given two numbers in registers r1 and r2. You may assume that these numbers are greater than 0. Write a SimpleRISC program to find out the LCM of these two numbers. Save the result in r0.

```
@ let the numbers be A(r1) and B(r2)
    mov r3, 1          @ idx = 1
    mov r4, r1         @ L = A
.loop:
    mod r5, r4, r2     @ L = A % B
    cmp r5, 0          @ compare mod with 0
    beq .lcm           @ LCM found (L is the LCM)
    add r3, r3, 1      @ increment idx
    mul r4, r1, r3      @ L = A * idx
    b .loop
.lcm:
    mov r0, r4         @ result is equal to L
```



- * Write a SimpleRisc assembly program to find the smallest number that is a **sum of two cubes in two different ways** → 1729

Modifiers

Let us now consider the problem of loading a 32 bit constant into a register. The following code snippet shows us how to load the constant 0xFB12CDEF

```
/* load the upper two bytes */  
mov r0, 0xFB12  
lsl r0, r0, 16
```

```
/* load the lower two bytes with 0x CD EF */  
mov r1, 0x CDEF  
lsl r1, r1, 16  
lsr r1, r1, 16 /* top 16 bits are zeros */
```

```
/* load all the four bytes */  
add r0, r0, r1
```


Modifiers

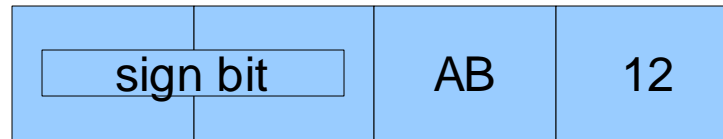
- * We can add the following modifiers to an instruction that has an immediate operand
- * Modifier :
 - * **default** : mov → treat the 16 bit immediate as a **signed number** (automatic sign extension)
 - * **(u)** : movu → treat the 16 bit immediate as an unsigned number
 - * **(h)** : movh → left shift the 16 bit immediate by 16 positions

Mechanism

- * The processor **internally converts** a 16 bit immediate to a 32 bit number
- * It uses **this 32 bit number** for all the computations
- * Valid only for arithmetic/logical insts
- * We can control the generation of this 32 bit number
 - * sign extension (**default**)
 - * treat the 16 bit number as unsigned (**u suffix**)
 - * load the 16 bit number in the upper bytes (**h suffix**)

More about Modifiers

- * default : `mov r1, 0xAB 12`



- * unsigned : `movu r1, 0xAB 12`



- * high: `movh r1, 0xAB 12`



Examples

- * Move : 0x FF FF A3 2B in r0

```
mov r0, 0xA32B
```

- * Move : 0x 00 00 A3 2B in r0

```
movu r0, 0xA32B
```

- * Move : 0x A3 2B 00 00 in r0

```
movh r0, 0xA32B
```

Example

* Set $r0 \leftarrow 0x\ 12\ AB\ A9\ 2D$

```
movh r0, 0x 12 AB  
addu r0, 0x A9 2D
```