Computer Systems Architecture

http://cs.nott.ac.uk/~txa/g51csa/

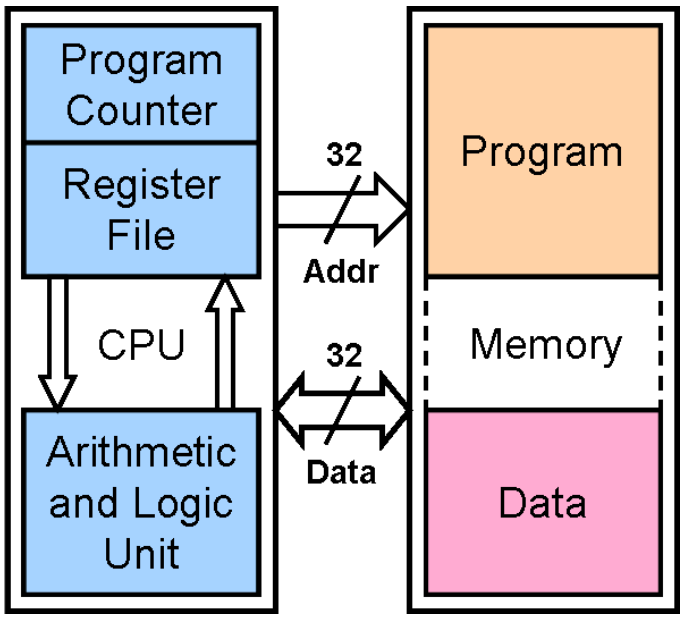
Thorsten Altenkirch

School of Computer Science and IT
University of Nottingham

Lecture 03: MIPS32, Arithmetic and System I/O

The University of Nottingham

Memory and Registers

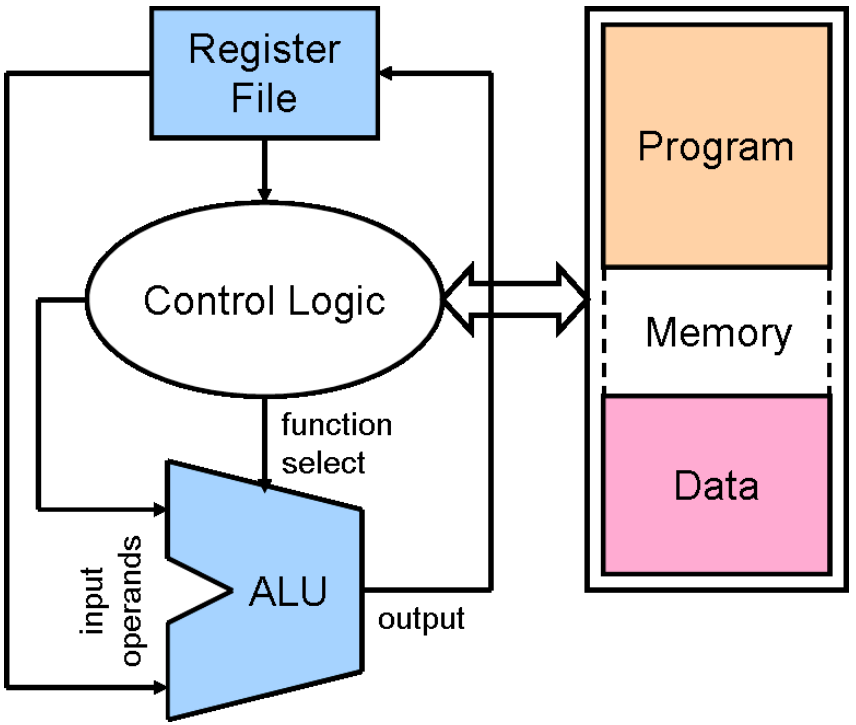


- Addresses are 32-bits words
  – 2^{32} different locations
- Words are 32-bits, or 4 bytes
  – 2^{30} addressable words
    - Word address must be aligned
- Registers are also word-sized
    - Only 32 general purpose
    - Special: `PC`, `Status`, `HI`, `LO`, ...
    - Floating point registers, ...

CPU Internals: Registers

Name	Number	Description
`$zero`	0	constant 0
`$at`	1	assembler temporary – do not use
`$v0`–`$v1`	2–3	function select; return result
`$a0`–`$a3`	4–7	function arguments
`$t0`–`$t9`	8–15, 24–25	temporaries
`$s0`–`$s7`	16–23	saved registers
`$k0`–`$k1`	26–27	kernel registers – do not use
`$gp`	28	global heap pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

CPU Overview



- Most instructions use ALU
- Function from instruction
- Two input, one output
- One input may be immediate
- Output always to register
  - Memory access orthogonal

Instruction Format

General Syntax

Three operands	Two operands	Other
`op dst, src, src`	`op dst, src`	`op`
`op dst, src, imm`	`op dst, imm`	`op src`

`op` operation code, or mnemonic

`dst` destination register

- always come before source operands

`src` source register

`imm` immediate value (16-bit)

- encoded in the instruction

The University of Nottingham

Arithmetic: Add

add dst, src_0, src_1

- Adds src_0 and src_1, placing the result in dst
- $dst := src_0 + src_1$

Example

Before

$s0 = 4	$t0 = 5	$t1 = 6

```
add $s0, $t0, $t1
```

After

$s0 = 11	$t0 = 5	$t1 = 6

Arithmetic: Subtract

`sub` dst, src_0, src_1

- Subtracts src_1 from src_0, placing the result in dst
- $dst := src_0 - src_1$

Example

Before

`$s0` = 4	`$t0` = 5	`$t1` = 6

```
add $s0, $t0, $t1
```

After

`$s0` = −1	`$t0` = 5	`$t1` = 6

The University of Nottingham

Arithmetic: Add Immediate

`addi` *dst*, *src*, *imm*

- Adds the 16-bit *imm* to *src*, placing the result in *dst*
- *dst* := *src* + *imm*

Example

Before

`$s0 = 4`	`$t0 = 5`	`$t1 = 6`

```
addi $s0, $t0, 37
```

After

`$s0 = 42`	`$t0 = 5`	`$t1 = 6`

Other arithmetic operations

Multiplication

`mul` *dst*, src_0, src_1

- Multiplies src_0 by src_1, placing the result in *dst*
- *dst* := src_0 * src_1

- The result may not fit into 32 bit.

Division

`div` *dst*, src_0, src_1

- Divides src_0 by src_1, placing the result in *dst*
- *dst* := src_0 / src_1

- We loose the remainder.

Example: A Short Calculation

Example

Before

$s0 = 3	$s1 = 4	$t0 = 5

```
add $s0, $s1, $s1
addi $s1, $s0, 6
sub $s0, $s1, $t0
```

After

$s0 = 9	$s1 = 14	$t0 = 5

Assignment

`move` *dst*, *src*

- Assign the value in *src* to *dst* — *dst* := *src*

There is no `move` instruction...

- Assembler translates *dst* := *src* as *dst* := *src* + 0:
  - `add` *dst*, *src*, `$zero`
- `move` is one of many *pseudoinstructions*
  - RISC philosophy of keeping features orthogonal

Load Immediate / Load Address

`li` *dst*, *imm* / `la` *dst*, **label**

- Loads constant *imm* into *dst* — *dst* := *imm*
- Loads address represented by *label* — *dst* := [*label*]

There is no `li` or `la` instruction...

- Instructions are 32 bits; *imm* may not fit!
- If *imm* representable with 16 bits:
  - `ori` *dst*, `$zero`, *imm*
- Otherwise, $imm = xxxxyyyy_{16}$:
  - `lui` *dst*, $xxxx_{16}$
    `ori` *dst*, *dst*, $yyyy_{16}$

The University of Nottingham

System Services

`syscall`

- Requests an operating system service
  - SPIM mimicks a simple OS
  - User input / output via the 'Console' window
- Function selected depends on `$v0`
- May destroy `$v0`–`$v1`, `$a0`–`$a3`, `$t0`–`$t9`, `$ra`
- But always preserves `$s0`–`$s7`

System Calls Reference

- Complete list in *Hennessey and Patterson*, Appendix A-44

Service	$v0	Arguments	Result
`print_int`	1	$a0	*none*
`print_string`	4	address in $a0	*none*
`read_int`	5	*none*	$v0
`read_string`	8	into $a0, $a1 length	*none*
`sbrk`	9	allocate $a0 bytes	starting at $v0
`exit`	10	*none*	*never!*

Example: Input, Output and Arithmetic

```
        .data
nl:     .asciiz "\n"
        .text
        .globl main
main:   li $v0, 5   # read_int
        syscall
        add $a0, $v0, $v0
        li $v0, 1   # print_int
        syscall
        la $a0, nl
        li $v0, 4   # print_string
        syscall
        j $ra
```

Reading. . .

- Read *Hennessey and Patterson*
  - §2.1–§2.2 for revision
  - §2.3 for information; essential parts will be covered later