Machine Architecture - Lecture 5

Word Address	Data				
⋮	⋮				⋮
0000000C	4 0	F 3	0 7	8 8	Word 3
00000008	0 1	E E	2 8	4 2	Word 2
00000004	F 2	F 1	A C	0 7	Word 1
00000000	A B	C D	E F	7 8	Word 0

Ioannis Ivrissimtzis

ioannis.ivrissimtzis@durham.ac.uk

MIPS – addressing modes

Slide material acknowledgements: Magnus Bordewich

MIPS addressing modes

$$\left.\begin{array}{l}\text{Register Only}\\ \\ \text{Immediate}\\ \\ \text{Base Addressing}\end{array}\right\}\ \text{Reading and writing operands}$$

$$\left.\begin{array}{l}\text{PC-Relative}\\ \\ \text{Pseudo-direct}\end{array}\right\}\ \text{Writing the Program Counter}$$

Register Only and immediate

Register Only

Uses registers for all source and destination operands.

R-type instructions use Register Only addressing.

Immediate

Uses registers and a 16-bit immediate as operands.

Some of the I-type instructions use Immediate addressing (depending on how the 16-bit immediate is used).

Base addressing

Used in memory access instructions.

Implemented by I-type instructions.

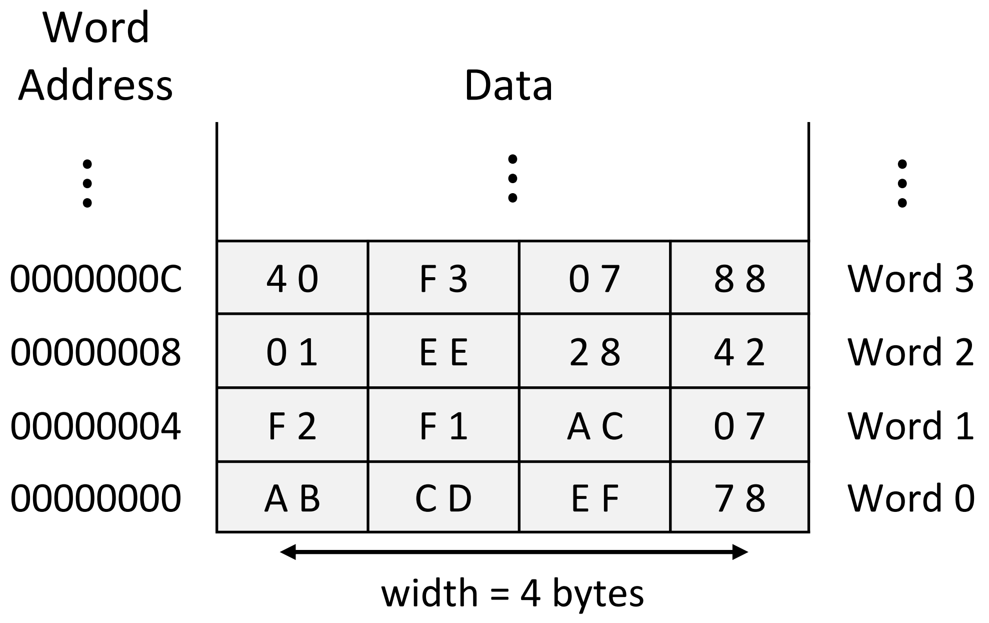
The address of the memory operand is computed by adding the base address in register `rs` to a 16-bit offset stored in `imm`.

Base addressing

word : 32-bits

byte : 8-bits

MIPS uses 32-bit memory addresses and memory is byte addressable.



Word Address	Data				
⋮	⋮				⋮
0000000C	4 0	F 3	0 7	8 8	Word 3
00000008	0 1	E E	2 8	4 2	Word 2
00000004	F 2	F 1	A C	0 7	Word 1
00000000	A B	C D	E F	7 8	Word 0

Base addressing

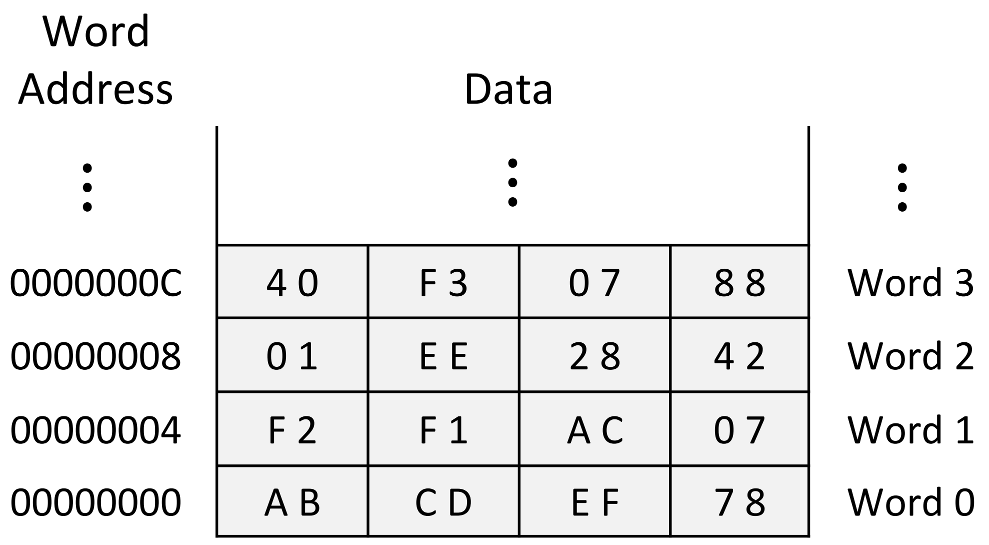
The instructions:

```
lw      # load word
lb      # load byte
sw      # store word
sb      # store byte
```

read and write data from and to the memory.

Durham University

Base addressing



```
lw $s0, 0($0)         # read data word 0 (0xABCDEF78 in the
                      # above example) and load it into $s0
```

Notice the assembly syntax for that I-Type instruction.

The Word Address 0 is the sum of the `imm` 0 and the value of register $0, which is always zero (look at the list of registers in the previous lecture).

Base addressing

Word Address	Data				
⋮		⋮			⋮
0000000C	4 0	F 3	0 7	8 8	Word 3
00000008	0 1	E E	2 8	4 2	Word 2
00000004	F 2	F 1	A C	0 7	Word 1
00000000	A B	C D	E F	7 8	Word 0

```
lw $s1, 8($0)        # read data Word 2 (0x01EE2842 in the
                     # above example) and load it into $s1
```

The Word Address 8 (corresponding to Word 2) is the sum of the `imm` 8 and the value of register $0, which is zero.

Base addressing

lw, sw .. blocking operation

```
sw $s3, 4($0)          # write $s3 to data Word 1

sw $s4, 0x20($0)       # write $s4 to data Word 8
                       # Notice the use of hexadecimal in the
                       # imm, which is supported by the MIPS
                       # assembly

sw $s5, 200($0)        # write $s5 to data Word 50
```

Base addressing

The assembly code

```
lw $t2, 32($0)
```

is translated to the machine language I-Type instruction

op	rs	rt	imm
35	0	10	32
100011	00000	01010	0000 0000 0010 0000
6 bits	5 bits	5 bits	16 bits

Base addressing

The assembly code

```
sw $s1, 4($t1)
```

is translated to the machine language I-Type instruction

`op`	`rs`	`rt`	`imm`
43	9	17	4
101011	01001	10001	0000 0000 0000 0100
6 bits	5 bits	5 bits	16 bits

PC-relative addressing

instruction finished
↓
next instruction
PC

Branching instructions use PC-relative addressing to specify the new value of the PC (Program Counter) if the branch is taken.

Consider the following MIPS Assembly code fragment:

```
0x40    loop:  add   $t1, $a0, $s0
0x44           lb    $t1, 0($t1)
0x48           add   $t2, $a1, $s0
0x4C           sb    $t1, 0($t2)
0x50           addi  $s0, $s0, 1
0x54           bne   $t1, $0, loop
0x58           lw    $s0, 0($sp)
```

label

Carry flag □ zero flag ☑

error flag □ overflow □

On the left hand side, in hexadecimal, is the word address where the instruction is stored, i.e., the PC value when the instruction is executed.

PC-relative addressing

```
0x40    loop:  add  $t1, $a0, $s0
0x44           lb   $t1, 0($t1)
0x48           add  $t2, $a1, $s0
0x4C           sb   $t1, 0($t2)
0x50           addi $s0, $s0, 1
0x54           bne  $t1, $0, loop
0x58           lw   $s0, 0($sp)
```

The branching instruction in this fragment is the I-Type `bne`.

It compares the values of registers `$t1` and `$0` and **b**ranches when they are **n**ot **e**qual.

The new value of the PC if the branch is taken is computed based on the immediate corresponding to the label `loop`.

PC-relative addressing

```
0x40    loop:  add   $t1, $a0, $s0
0x44           lb    $t1, 0($t1)
0x48           add   $t2, $a1, $s0
0x4C           sb    $t1, 0($t2)
0x50           addi  $s0, $s0, 1
0x54           bne   $t1, $0, loop
0x58           lw    $s0, 0($sp)
```

Notice that by labelling some instructions (here `loop`), if we use assembly, we do not have to worry how the new PC value will be computed.

The issue here is not how to write assembly, but how assembly code is translated to machine language.

PC-relative addressing

```
0x40    loop:  add  $t1, $a0, $s0
0x44           lb   $t1, 0($t1)
0x48           add  $t2, $a1, $s0
0x4C           sb   $t1, 0($t2)
0x50           addi $s0, $s0, 1
0x54           bne  $t1, $0, loop
0x58           lw   $s0, 0($sp)
```

To calculate the `imm`, we take the PC value immediately after the branching instruction, here 0x58, we subtract it from the Branch Target Address (BTA), here 0x40, and divide by 4. Here, result will be -6.

Equivalently, we count the number of instructions from PC+4 to BTA, using a minus sign if BTA is above PC+4.

PC-relative addressing

The assembly code

```
bne  $t1, $0, loop
```

is translated to the machine language I-Type instruction

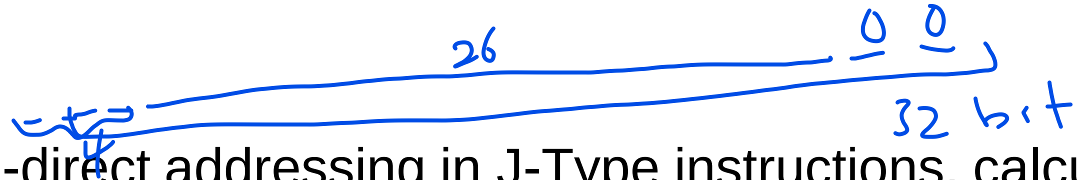
op	rs	rt	imm
5	9	0	-6
000101	01001	00000	1111 1111 1111 1010
6 bits	5 bits	5 bits	16 bits

Notice that the 16-bits of the `imm`, field are used to represent integers from −32,768 to 32,767 rather than form 0 to 65,535.

Pseudo-direct addressing

In direct addressing, an address is specified in the instruction.

MIPS cannot support direct addressing because we would need 32-bit for the address and 6-bits for the opcode, while the instruction has 32 bits only.



MIPS uses pseudo-direct addressing in J-Type instructions, calculating the new value of the PC, called Jump Target Address (JTA), as follows:

- The two least significant bits are set to 0 (instructions are word aligned and word addresses are multiples of 4).
- The next 26 bits are taken from the `addr` field of the J-Type instruction.
- The four most significant bits are taken from the current PC+4 value.

Pseudo-direct addressing

```
0x0040005C          j  sum
...
...
0x004000A0   sum: add $v0, $a0, $a1
```

+4

To find the value of the `addr` field for the `j` instruction, we first find

PC+4 = 0x0040005C = 0x00400060 = 0000 0000 0100 0000 0000 0000 0110 0000 $_2$

The 4 most significant bits of PC+4 are 0000. We should have

0000 xxxx xxxx xxxx xxxx xxxx xxxx xx00 = 0x004000A0

where the x's are the bits in the `addr` field. In binary

0000 xxxx xxxx xxxx xxxx xxxx xxxx xx00 = 0000 0000 0100 0000 0000 0000 1010 0000

giving

`addr` = 0000 0100 0000 0000 0000 1010 00

Pseudo-direct addressing

```
0x0040005C          j  sum
...
...
0x004000A0   sum: add $v0, $a0, $a1
```

The assembly code

`j   sum`

is translated to the machine language J-Type instruction

op	addr
000101	00000100000000000000101000
6 bits	26 bits

Note: The reuse of the four most significant bits of the current PC+4 in the calculation of the JTA limits the range of the instruction `j`.