

CS2100 Tutorial 4

Datapath

(Note: I'm very sick so another TA will teach this class with (most likely) not this set of slides)

Overview

Q1) Counting number of instructions

Q2) Pseudo-instructions implementation

Q3) Encoding MIPS instructions

Q4) MIPS Datapath

Q5) (Extra) GPU vs CPU

Q1. Number of Instructions

An ISA has 16-bit instructions and 5-bit addresses; two classes of instructions: A has one address; B has two addresses.

Both classes exist, and encoding space is fully utilized.

Bonus:

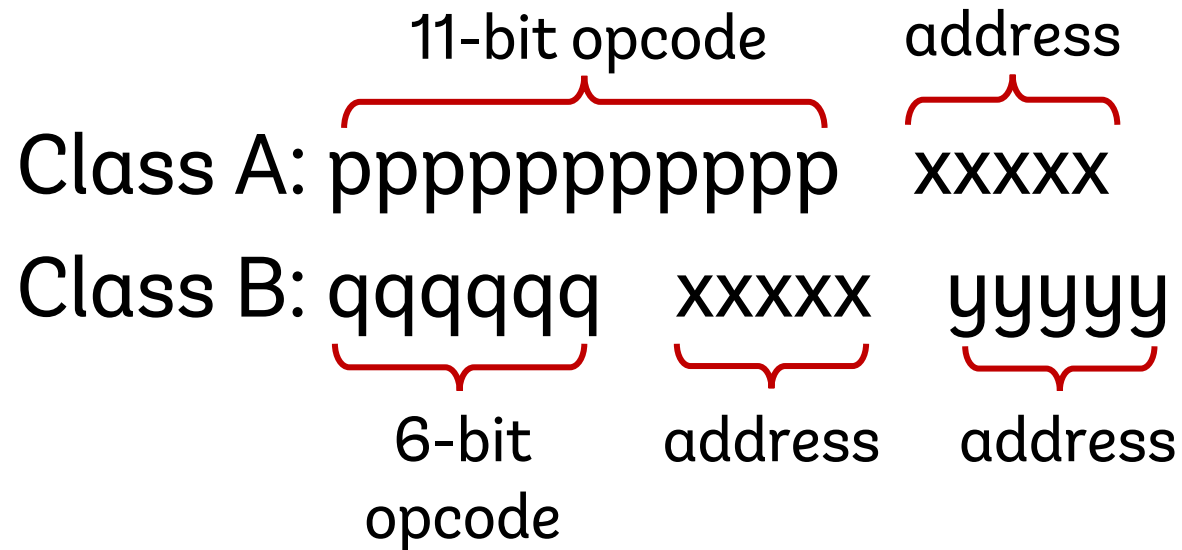
- What happens if we don't stipulate both classes existing?
- What happens if we don't need the encoding space to be fully utilized?

Q1. Number of Instructions

An ISA has 16-bit instructions and 5-bit addresses; two classes of instructions: A has one address; B has two addresses.

Both classes exist, and encoding space is fully utilized.

(a) What is the minimum total number of instructions?

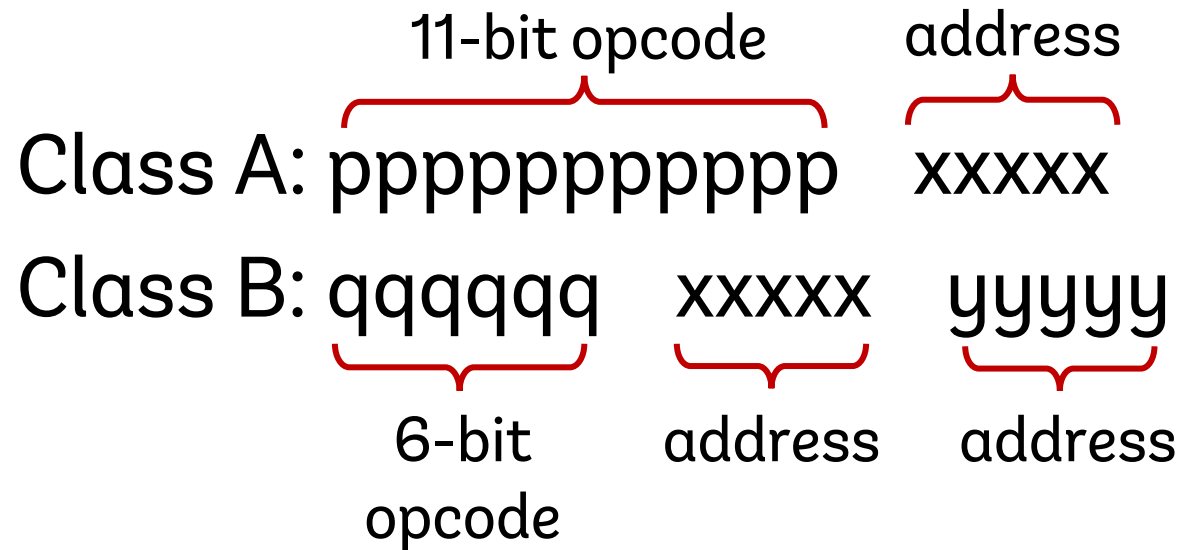


Q1. Number of Instructions

An ISA has 16-bit instructions and 5-bit addresses; two classes of instructions: A has one address; B has two addresses.

Both classes exist, and encoding space is fully utilized.

(a) What is the minimum total number of instructions?



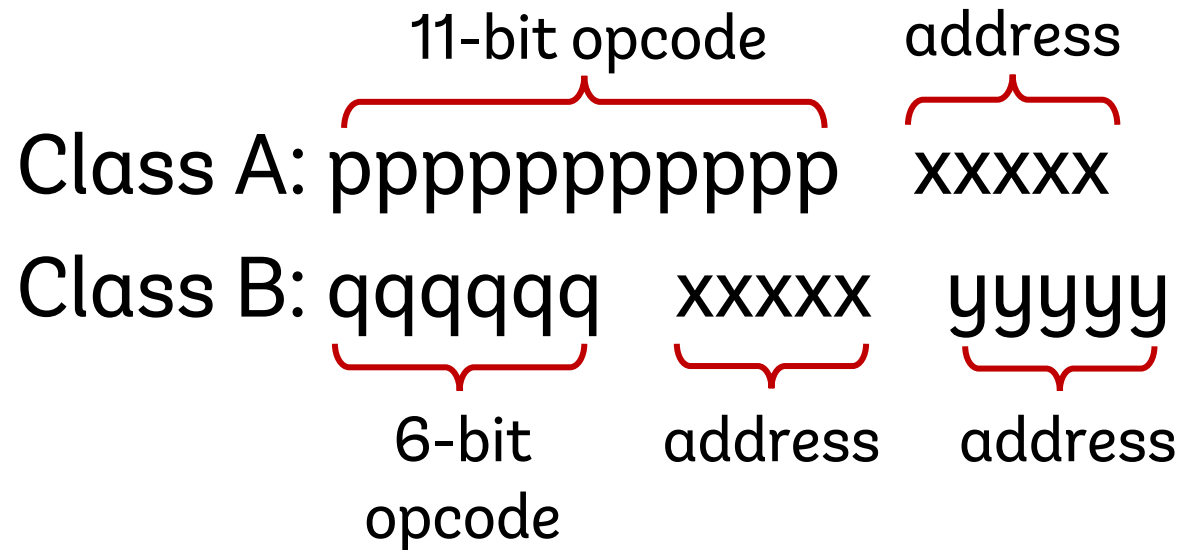
Answer: 95

Q1. Number of Instructions

An ISA has 16-bit instructions and 5-bit addresses; two classes of instructions: A has one address; B has two addresses.

Both classes exist, and encoding space is fully utilized.

(a) What is the maximum total number of instructions?

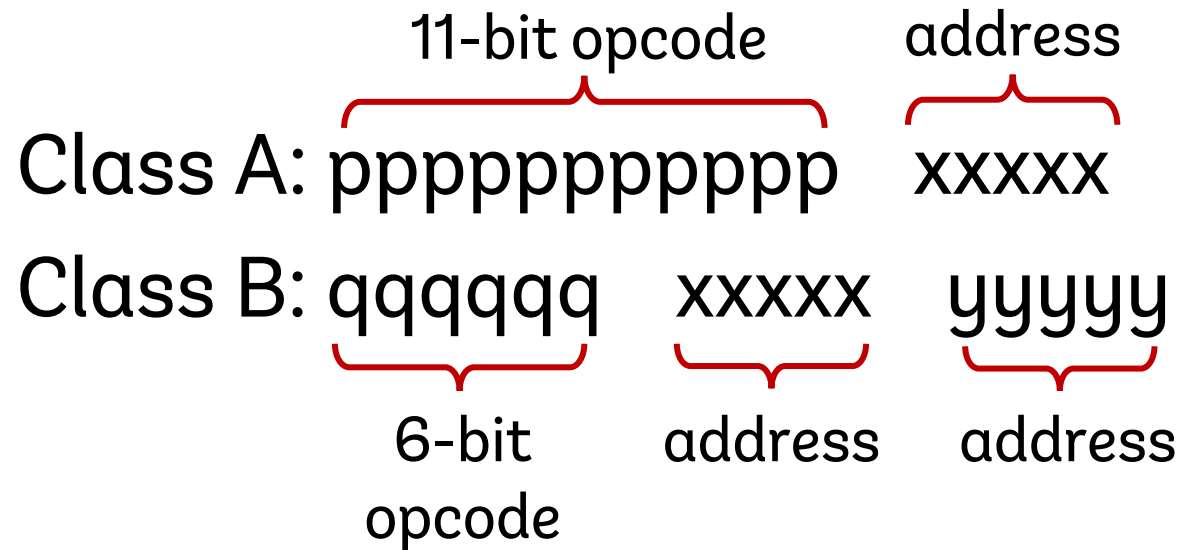


Q1. Number of Instructions

An ISA has 16-bit instructions and 5-bit addresses; two classes of instructions: A has one address; B has two addresses.

Both classes exist, and encoding space is fully utilized.

(a) What is the maximum total number of instructions?



Answer: 2017

Q2. Pseudo-instruction implementation

- Several pseudo-instructions available
 - Check the MIPS sheet!
 - Can't use in exam, except when explicitly allowed
- Question: how is it actually implemented?

Q2. Pseudo-instruction implementation

(a) `bgt $r1, $r2, L`

[Live demo: QtSpim]

Note:

We can't use `$r1` and `$r2` in QtSpim!

Substitute with other registers, e.g. `$t0`, `$t1`

Note 2:

Keep `$r1`, `$r2`, and `L` as-is in your final answer.

Don't change it to other register / label names.

Q2. Pseudo-instruction implementation

(a) `bgt $r1, $r2, L`
`= blt $r2, $r1, L`

Note:

While `$at` is used in QtSpim, we will consider your answer correct with any other register.

User Text Segment [00400000]..[00440000]
[00400000] 0128082a `slt $1, $9, $8` ; 1: `bgt $t0, $t1, e`
[00400004] 14200001 `bne $1, $0, 4 [e-0x00400004]`

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
\$zero	0	The Constant Value 0	N.A.
\$at	1	Assembler Temporary	No
\$v0-\$v1	2-3	Values for Function Results and Expression Evaluation	No
\$a0-\$a3	4-7	Arguments	No
\$t0-\$t7	8-15	Temporaries	No
\$s0-\$s7	16-23	Saved Temporaries	Yes
\$t8-\$t9	24-25	Temporaries	No
\$k0-\$k1	26-27	Reserved for OS Kernel	No
\$gp	28	Global Pointer	Yes
\$sp	29	Stack Pointer	Yes
\$fp	30	Frame Pointer	Yes
\$ra	31	Return Address	No

`$1` is `$at` (Assembler Temporary)

Answer:

`slt $at, $r2, $r1`
`bne $at, $zero, L`

Q2. Pseudo-instruction implementation

(b) `bge $r1, $r2, L`

Q2. Pseudo-instruction implementation

(b) `bge $r1, $r2, L`

Idea: `if($r1 >= r2) ==> if(!($r1 < $r2))`

Q2. Pseudo-instruction implementation

(b) bge \$r1, \$r2, L

Idea: `if($r1 >= r2) ==> if(!($r1 < $r2))`

From (a):

`blt $r1, $r2, L`

is equivalent to

`slt $at, $r1, $r2`

`bne $at, $zero, L`

Q2. Pseudo-instruction implementation

(b) bge \$r1, \$r2, L

Idea: `if($r1 >= r2) ==> if(!($r1 < $r2))`

From (a):

`blt $r1, $r2, L`

is equivalent to

`slt $at, $r1, $r2`

`bne $at, $zero, L`

Idea 2: "negating" bne becomes beq

Answer:

`slt $at, $r2, $r1`

`beq $at, $zero, L`

Q2. Pseudo-instruction implementation

(c) `ble $r1, $r2, L`

Q2. Pseudo-instruction implementation

(c) `ble $r1, $r2, L`

Idea:

`if ($r1 < $r2)`

`=> if ($r2 >= $r1)`

`=> if (!($r2 < $r1))`

Q2. Pseudo-instruction implementation

(c) `ble $r1, $r2, L`

Idea:

`if ($r1 < $r2)`

`=> if ($r2 >= $r1)`

`=> if (!($r2 < $r1))`

From (a):

`blt $r2, $r1, L`

is equivalent to

`slt $at, $r2, $r1`

`bne $at, $zero, L`

"negating" `bne` becomes `beq`

Answer:

`slt $at, $r1, $r2`

`beq $at, $zero, L`

Q2. Pseudo-instruction implementation

(d) `li $r, imm`

Thing to note: imm here can be any 32-bit number

Q2. Pseudo-instruction implementation

(d) `li $r, imm`

Thing to note: `imm` here can be any 32-bit number

Scenario 1: the number is 0-65535 (fits within 16 bits)

Answer:

`ori $r, $zero, imm`

Q2. Pseudo-instruction implementation

(d) `li $r, imm`

Thing to note: imm here can be any 32-bit number

Scenario 2: the number can't fit within 16 bits

Answer: `lui + ori` (see: previous tutorial)

Note:

The assembler will parse the imm value.

Try the following:

```
li $t0, 2100
```

```
li $t1, 4294967295
```

```
li $t2, -1
```

Q2. Pseudo-instruction implementation

(d) nop

Q2. Pseudo-instruction implementation

(d) nop

Answer: 0000 0000 0000 0000 0000 0000 0000 0000

Q2. Pseudo-instruction implementation

(d) nop


Answer: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
opcode funct

See MIPS reference sheet: this is an sll instruction

Q2. Pseudo-instruction implementation

(d) nop

Answer: 0000 0000 0000 0000 0000 0000 0000 0000



The diagram shows the MIPS instruction format for the nop instruction. The 32-bit instruction is divided into four 8-bit fields: rs (red box), rt (orange box), rd (yellow box), and shamt (purple box). Each field contains four zeros. The labels rs, rt, rd, and shamt are placed below their respective fields in matching colors.

rs rt rd shamt

See MIPS reference sheet again: sll \$zero, \$zero, 0

Note:

Why do we need a nop operation?

Answer in pipelining section of the course

Q3. MIPS Encoding

(a) `beq $1, $3, 0x12`

Q3. MIPS Encoding

(a) `beq $1, $3, 0x12`

Answer: `0x1023000C`

Q3. MIPS Encoding

(b) lw \$24, 0(\$15)

Q3. MIPS Encoding

(b) `lw $24, 0($15)`

Answer: `0x8DF80000`

Q3. MIPS Encoding

(c) `sub $25, $20, 0x5`

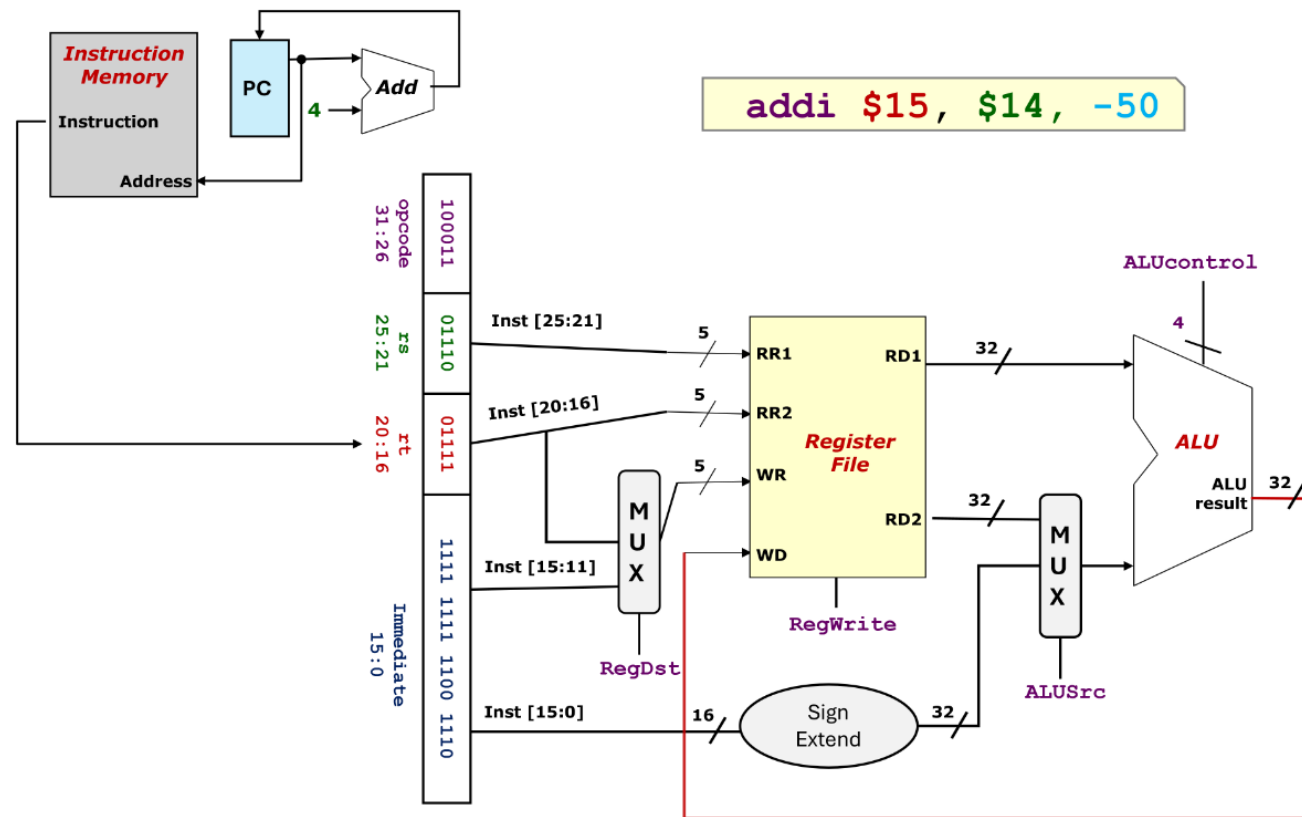
Q3. MIPS Encoding

(c) `sub $25, $20, 0x5`

Answer: `0x0285C822`

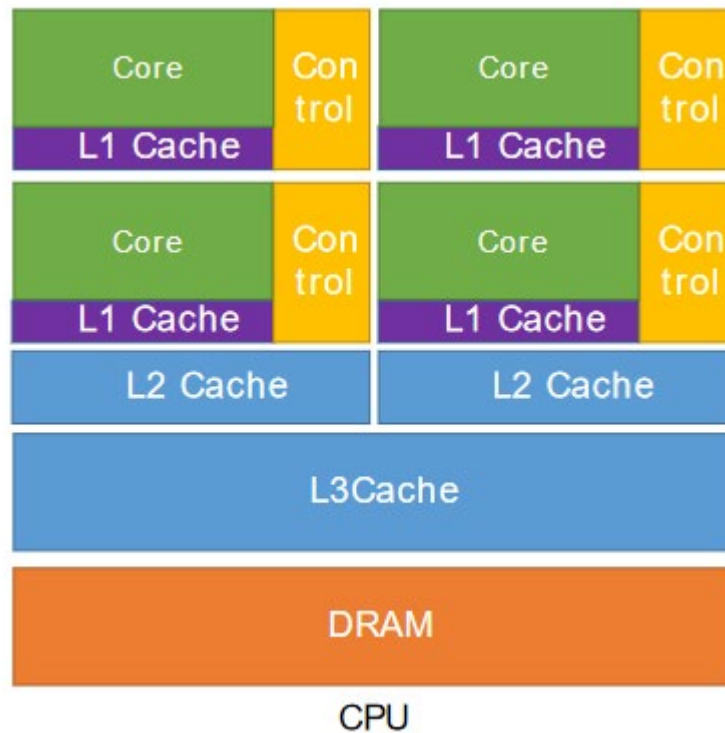
Q4. MIPS Datapath

Q: Draw the MIPS datapath for `addi $15, $14, -50`



Q5. CPUs vs GPUs

- Architecture-wise, not much difference!
 - Still Fetch-Decode-Execute cycle
 - Difference: multiple cores can have the same datapath for fetch+decode



Q5. CPUs vs GPUs

- Architecture-wise, not much difference!
- Usage difference:
 - CPU can handle complex tasks, but has less parallel capabilities
 - Your machine probably has somewhere between 8 to 64 cores
 - Try the 'lstopo --of ascii' command to see nice graphics
 - GPU handles simpler workloads but massively more parallel
 - 10+k cores is not unheard of. (Image source: <https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/>)
 - Each group of 32/64 cores does the same thing on different data – each core have their own register

Specs										
	GeForce RTX 3090 Ti	GeForce RTX 3090	GeForce RTX 3080 Ti	GeForce RTX 3080	GeForce RTX 3070 Ti	GeForce RTX 3070	GeForce RTX 3060 Ti	GeForce RTX 3060	GeForce RTX 3050 (8 GB)	GeForce RTX 3050 (6 GB)
NVIDIA CUDA Cores	10752	10496	10240	8960 / 8704	6144	5888	4864	3584	2560 ¹⁸	2304

End of Tutorial 4

- Slides uploaded on github.com/theodoreleebrant/TA-2425S1
- Email: theo@comp.nus.edu.sg
- Anonymous feedback:
bit.ly/feedback-theodore
(or scan on the right)

