CS2100 Tutorial 3

MIPS

(Take the Reference Sheet in front)

Will start at :05 as usual

Slides by Theodore, adapted from Prof. Aaron's slides

Admin stuff

- Discussion question:
  - do on your own, discuss in Canvas, ask me if anything
- MIPS Sheet
  - take one (in-front) if you haven't last week
  - would be useful until exam time

Overview

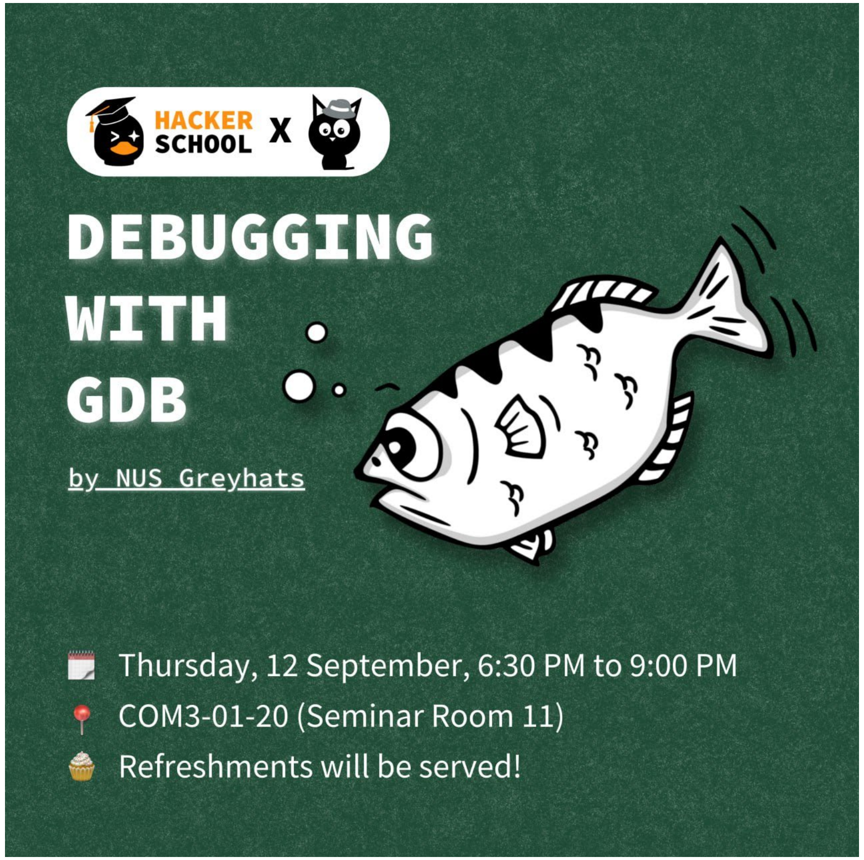
Q1) C to MIPS translation

Q2) MIPS instruction encoding

Q3) Completing MIPS code

Bonus (look on your own): Next steps for C

Promotion: "Debugging with GDB" talk



RSVP:

NUS: https://nus.campuslabs.com/engage/event/10352925

Non-NUS: https://docs.google.com/forms/d/e/1FAIpQLScJpX-bDTsPgGmmCrmmpR6HCTdLU7UbOCno523ZJkSPUFd4VQ/viewform

Ever wondered how to incorporate debugging with GDB into your C programming workflow? Had a hard time learning to use GDB to debug your application? In this workshop, we will be learning to use GDB to effectively debug our application.

Requirements:
If you're using macOS or Linux, you're all set!
Windows users: You can install Windows Subsystem for Linux (WSL) or run Ubuntu in VirtualBox.

Speaker Profile:
NUS Greyhats are an information security interest group based in the National University of Singapore. They play CTFs and organize weekly meetups they call Security Wednesdays!

Prelude to Q1: C’s “short-circuit eval”

(If you took CS1101S, this should be familiar to you!)

Q: What’s the result of evaluating the following in C:

```
int main() {
    if (1 > 2 && 5/0 > 3) {
        print(“aye”);
    } else {
        print(“nay”);
    }
}
```

```
A) Compile Error
B) Segmentation fault
C) “aye”
D) “nay”
E) None of the above
```

C short-circuit evaluation (1)

0

```
int main() {
    if (1 > 2 && 5/0 > 3) {
        print(“aye”);
    } else {
        print(“nay”);
    }
}
```

Compile Error — 0%

Segmentation fault — 0%

"aye" — 0%

"nay" — 0%

None of the above — 0%

Prelude to Q1: C’s “short-circuit eval”

(If you took CS1101S, this should be familiar to you!)

Q: If we switched the order of the operands…?

```
int main() {
    if (5/0 > 3 && 1 > 2) {
        print(“aye”);
    } else {
        print(“nay”);
    }
}
```

```
A) Compile Error
B) Segmentation fault
C) “aye”
D) “nay”
E) None of the above
```

C short-circuit evaluation (2)

0

```
int main() {
    if (5/0 > 3 && 1 > 2) {
        print("aye");
    } else {
        print("nay");
    }
}
```

Compile Error	0%
Segmentation fault	0%
"aye"	0%
"nay"	0%
None of the above	0%

Q1: C to MIPS

```
char str[size] = { ... }; // some string
int lo, hi, matched; // matched = 1 (palindrome);
                     // matched = 0 (not palindrome)
// Translate to MIPS from this point onwards
lo = 0;
hi = size-1;
matched = 1; // assume this is a palindrome
while ((lo < hi) && matched) {
        if (str[lo] != str[hi])
                matched = 0; // found a mismatch
        else {
                lo++;
                hi--;
}}
```

Variable mappings:
- lo → $s0;
- hi → $s1;
- matched → $s3;
- Base address of str[] → $s4;
- size → $s5

Q1. C → MIPS

```
lo = 0; hi = size-1; matched = 1;
while ((lo < hi) && matched) {
  if (str[lo] != str[hi]) matched = 0;
    else { lo++; hi--;
  }
}
```

```
        addi $s0, $zero, 0      # lo = 0
        addi $s1, $s5, -1       # hi = size – 1
        addi $s3, $zero, 1      # matched = 1
loop:   slt  $t0, $s0, $s1      # (lo < hi)?
        beq  $t0, $zero, exit   # exit if (lo >= hi)
        beq  $s3, $zero, exit   # exit if (match == 0)
        add  $t1, $s4, $s0      # addr of str[lo]
        lb   $t2, 0($t1)        # $t2 = str[lo]
        add  $t3, $s4, $s1      # addr of str[hi]
        lb   $t4, 0($t3)        # $t4 = str[hi]
        beq  $t2, $t4, else     # compare str[lo], str[hi]
        addi $s3, $zero, 0      # matched = 0
        j    loop               # can also be “j endW”
else:   addi $s0, $s0, 1        # lo++
        addi $s1, $s1, -1       # hi--
endW:   j loop
exit:
```

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Q1. C → MIPS

```
lo = 0; hi = size-1; matched = 1;
while ((lo < hi) && matched) {
  if (str[lo] != str[hi]) matched = 0;
  else { lo++; hi--; }
}
```

```
        addi $s0, $zero, 0         # lo = 0
        addi $s1, $s5, -1          # hi = size – 1
        addi $s3, $zero, 1         # matched = 1
loop:   slt  $t0, $s0, $s1         # (lo < hi)?
        beq  $t0, $zero, exit      # exit if (lo >= hi)
        beq  $s3, $zero, exit      # exit if (match == 0)
        add  $t1, $s4, $s0         # addr of str[lo]
        lb   $t2, 0($t1)           # $t2 = str[lo]
        add  $t3, $s4, $s1         # addr of str[hi]
        lb   $t4, 0($t3)           # $t4 = str[hi]
        beq  $t2, $t4, else        # compare str[lo], str[hi]
        addi $s3, $zero, 0         # matched = 0
        j    loop                  # can also be “j endW”
else:   addi $s0, $s0, 1           # lo++
        addi $s1, $s1, -1          # hi--
endW:   j loop
exit:
```

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Q1. C → MIPS

```
lo = 0; hi = size-1; matched = 1;
while ((lo < hi) && matched) {
  if (str[lo] != str[hi]) matched = 0;
    else { lo++; hi--;
  }
}
```

```
        addi $s0, $zero, 0        # lo = 0
        addi $s1, $s5, -1         # hi = size – 1
        addi $s3, $zero, 1
loop:   slt  $t0, $s0, $s1        # (lo < hi)?
        beq  $t0, $zero, exit     # exit if (lo >= hi)
        beq  $s3, $zero, exit     # exit if (match == 0)
        add  $t1, $s4, $s0        # addr of str[lo]
        lb   $t2, 0($t1)          # $t2 = str[lo]
        add  $t3, $s4, $s1        # addr of str[hi]
        lb   $t4, 0($t3)          # $t4 = str[hi]
        beq  $t2, $t4, else       # compare str[lo], str[hi]
        addi $s3, $zero, 0        # matched = 0
        j    loop                 # can also be "j endW"
else:   addi $s0, $s0, 1          # lo++
        addi $s1, $s1, -1         # hi--
endW:   j loop
exit:
```

Note 1: `blt` **pseudo-instruction**

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Q1. C → MIPS

```
lo = 0; hi = size-1; matched = 1;
while ((lo < hi) && matched) {
  if (str[lo] != str[hi]) matched = 0;
    else { lo++; hi--;
  }
}
```

```
        addi $s0, $zero, 0      # lo = 0
        addi $s1, $s5, -1       # hi = size – 1
        addi $s3, $zero, 1      # matched = 1
loop:   slt  $t0, $s0, $s1      # (lo < hi)?
        beq  $t0, $zero, exit   # exit if (lo >= hi)
        beq  $s3, $zero, exit   # ex
        add  $t1, $s4, $s0      # addr of str[lo]
        lb   $t2, 0($t1)        # $t2 = str[lo]
        add  $t3, $s4, $s1      # addr of str[hi]
        lb   $t4, 0($t3)        # $t4 = str[hi]
        beq  $t2, $t4, else     # compare str[lo], str[hi]
        addi $s3, $zero, 0      # matched = 0
        j    loop               # can also be “j endW”
else:   addi $s0, $s0, 1        # lo++
        addi $s1, $s1, -1       # hi--
endW:   j loop
exit:
```

Note 2: how to load

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Q1. C → MIPS

```
lo = 0; hi = size-1; matched = 1;
while ((lo < hi) && matched) {
  if (str[lo] != str[hi]) matched = 0;
    else { lo++; hi--;
  }
}
```

```
        addi $s0, $zero, 0       # lo = 0
        addi $s1, $s5, -1        # hi = size – 1
        addi $s3, $zero, 1       # matched = 1
loop:   slt  $t0, $s0, $s1       # (lo < hi)?
        beq  $t0, $zero, exit    # exit if (lo >= hi)
        beq  $s3, $zero, exit    # exit i
        add  $t1, $s4, $s0       # addr of str[lo]
        lb   $t2, 0($t1)         # $t2 = str[lo]
        add  $t3, $s4, $s1       # addr of str[hi]
        lb   $t4, 0($t3)         # $t4 = str[hi]
        beq  $t2, $t4, else      # compare str[lo], str[hi]
        addi $s3, $zero, 0       # matched = 0
        j    loop                # can also be "j endW"
else:   addi $s0, $s0, 1         # lo++
        addi $s1, $s1, -1        # hi--
endW:   j loop
exit:
```

Note 3: lb vs lw

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Q1b.

```
        addi $s0, $zero, 0
        addi $s1, $s5, -1
        addi $s3, $zero, 1
loop:   slt  $t0, $s0, $s1          # (lo < hi)?
        beq  $t0, $zero, exit       # exit if (lo >= hi)
        beq  $s3, $zero, exit       # exit if (match == 0)
        add  $t1, $s4, $s0          # addr of str[lo]
        lb   $t2, 0($t1)            # $t2 = str[lo]
        add  $t3, $s4, $s1          # addr of str[hi]
        lb   $t4, 0($t3)            # $t4 = str[hi]
        beq  $t2, $t4, else         # compare str[lo], str[hi]
        addi $s3, $zero, 0          # matched = 0
        j    loop                   # can also be “j endW”
else:   addi $s0, $s0, 1            # lo++
        addi $s1, $s1, -1           # hi--
endW:   j loop
exit:
```

```
       add  $t1, $s4, $s0 # addr. of str[lo]
       add  $t3, $s4, $s1 # addr. of str[hi]
loop:  slt  $t0, $t1, $t3 # compare lo and hi addr
```

```
addi $t1, $t1, 1  # lo addr increment
addi $t3, $t3, -1 # hi addr decrement
```

Variable mappings: lo → $s0; hi → $s1; matched → $s3;
Base address of str[] → $s4; size → $s5

Prelude to Q2: MIPS Instruction Encoding

- Remember and distinguish between R/I/J instructions!
  - R-format instructions: operations using 3 registers OR shifts
    - Example: add, and, slt, sll
  - I-format instructions: operations using 2 register and an imm value
    - Example: addi, beq, lw, sw
  - J-format instruction: jump instructions
    - Example: j, jal
  - Other formats not in syllabus
    - See reference data: there's FI/FR-formats for floats
    - Assessments can have custom instruction formats!

Prelude to Q2: MIPS Instruction Encoding

- Be careful about the ordering!

Instruction:

`add`	`$t0,`	`$t1,`	`$t2`
(opcode 0, funct 20_{16})	rd (dest)	rs (source)	rt (target)

MIPS Reference Sheet:

R[rd] = R[rs] + R[rt]

	opcode	rs	rt	rd	shamt	funct
R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0

Q2: MIPS Instruction Encoding

```
addi $s1, $zero, 0
```

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}

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

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Q2: MIPS Instruction Encoding

```
addi $s1, $zero, 0
          rt         rs       SignExtImm
```

```
rt = $s1 = $(16+1) = $17
rs = $zero = $0
```

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)		OPCODE/FUNCT (Hex)
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}

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

BASIC INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	address				

Q2: MIPS Instruction Encoding

```
addi $s1, $zero, 0
     rt   rs     SignExtImm
```

```
rt = $s1 = $(16+1) = $17
rs = $zero = $0
```

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}

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

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Q2: MIPS Instruction Encoding

```
addi $s1, $zero, 0
          rt        rs        SignExtImm
```

```
rt = $s1 = $(16+1) = $17
rs = $zero = $0
```

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}

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

BASIC INSTRUCTION FORMATS

R	opcode (31–26)	rs (25–21)	rt (20–16)	rd (15–11)	shamt (10–6)	funct (5–0)
I	opcode (31–26)	rs (25–21)	rt (20–16)	immediate (15–0)		
J	opcode (31–26)	address (25–0)				

Ans: $(2011\ 0000)_{16}$

Q2: MIPS Instruction Encoding

0x11000002

=0b 0001 0001 0000 0000 0000 00000 0000 0010

First, check the opcode (first 6 bits), which is $000100_2 = 4_{16}$

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Q2: MIPS Instruction Encoding

```
0x11000002
=0b 0001 0001 0000 0000 0000 00000 0000 0010
```

Second, check the Reference Sheet for opcode 4_{16}.

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr*4	(4)	4_{hex}

Now we know it's a beq instruction with I-format.

Q2: MIPS Instruction Encoding

`0x11000002`

`=0b 0001 0001 0000 0000 0000 00000 0000 0010`

BASIC INSTRUCTION FORMATS

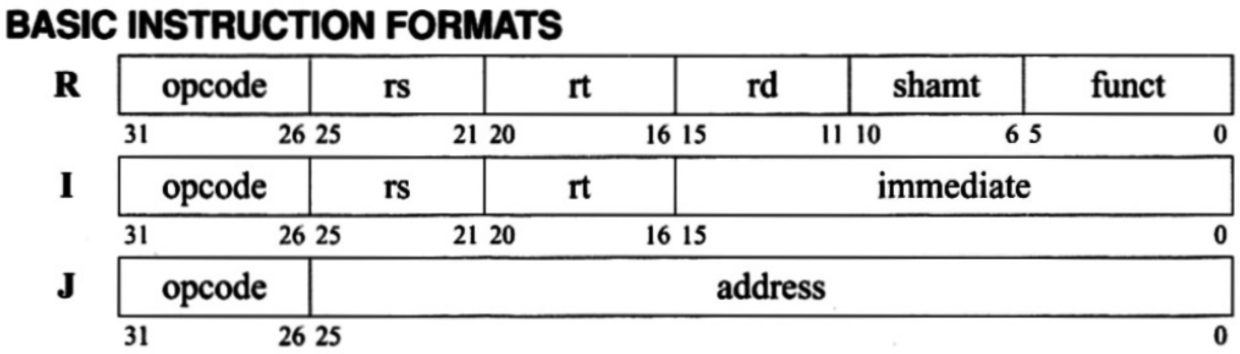
R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

Q2: MIPS Instruction Encoding

0x11000002

=0b 0001 0001 0000 0000 0000 00000 0000 0010

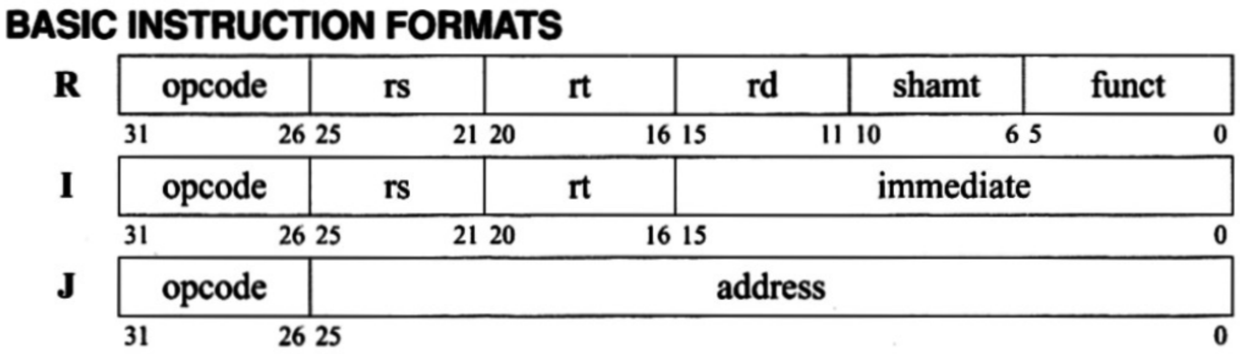
```
opcode   000100 = 4
rs        01000 = 8
rt        00000 = 0
immediate  0100 = 2
```



Q2: MIPS Instruction Encoding

```
0x11000002
=0b 0001 0001 0000 0000 0000 00000 0000 0010
```

```
opcode   000100 = 4
rs        01000 = 8
rt        00000 = 0
immediate 0100 = 2
```



\$8 is \$t0; \$0 is \$zero, and the immediate value is +2.

Q2: MIPS Instruction Encoding

```
0x11000002
=0b 0001 0001 0000 0000 0000 00000 0000 0010

opcode    --> beq
rs        --> $t0
rt        --> $zero
immediate --> +2
```

Q2: MIPS Instruction Encoding

```
0x11000002
=0b 0001 0001 0000 0000 0000 00000 0000 0010
```

```
opcode    --> beq
rs        --> $t0
rt        --> $zero
immediate --> +2
```

Next instr.

+2 instr.

	MIPS code
	addi $s1, $zero, 0
0x00084042	Loop: srl $t0, $t0, 1
0x11000002	
0x22310001	
	j loop
	exit:

Ans: beq $t0, $zero, exit

Q2: MIPS Instruction Encoding

`0x22310001`

`=0b 0010 0010 0011 0001 0000 0000 0000 0001`

opcode $010000_2 = 8_{16}$

This means `addi` instruction, I-format

Q2: MIPS Instruction Encoding

`0x22310001`

`=0b 0010 0010 0011 0001 0000 0000 0000 0001`

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}

opcode $010000_2 = 8_{16}$

This means `addi` instruction, I-format

`0b 0010 0010 0011 0001 0000 0000 0000 0001`

`opcode rs rt imm (SignExtImm)`

	opcode	rs	rt	immediate
I	31–26	25–21	20–16	15–0

Q2: MIPS Instruction Encoding

0x22310001

=0b 0010 0010 0011 0001 0000 0000 0000 0001

opcode $010000_2 = 8_{16}$

This means `addi` instruction, I-format

	opcode	rs	rt	imm (SignExtImm)
0b	0010 00	10 0011	1 0001	0000 0000 0000 0001
	8_{16}	17_{10}	17_{10}	1_{10}
		\$s1	\$s1	

Q2: MIPS Instruction Encoding

`0x22310001`

`=0b 0010 0010 0011 0001 0000 0000 0000 0001`

opcode $010000_2 = 8_{16}$

This means `addi` instruction, I-format

`0b`	`0010 00`	`10 0011`	`1 0001`	`0000 0000 0000 0001`
	opcode	rs	rt	imm (SignExtImm)
	8_{16}	17_{10}	17_{10}	1_{10}
		`$s1`	`$s1`	

Answer:

addi	**\$s1,**	**\$s1,**	**1**
inst	rt	rs	imm

Q2: MIPS Instruction Encoding

`j loop`

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)	OPCODE/FUNCT (Hex)
Jump	j	J	PC=JumpAddr	(5) 2_{hex}

J	opcode	address
	31 26	25 0

We first try to find the address of where we want to jump

`0x00400028 + 4`

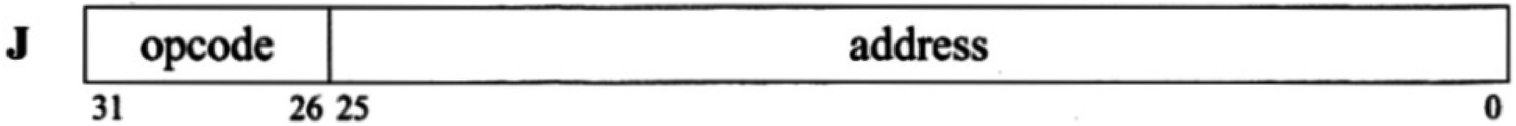
`= 0x0040002C`

MIPS code	Address
addi $s1, $zero, 0	0x00400028
Loop: srl $t0, $t0, 1	?
j loop	
exit:	

Q2: MIPS Instruction Encoding

`j loop`

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)	OPCODE/FUNCT (Hex)
Jump	j	J	PC=JumpAddr	(5) 2_{hex}



We first try to find the address of where we want to jump

`= 0x0040002C`

`= 0000 0000 0100 0000 0000 0000 0010 1100`

As everything is word-aligned, we remove the last 2 bits;

To make it fit, we need to also throw away the first 4 bits.

Q2: MIPS Instruction Encoding

```
j loop
```

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)	OPCODE/ FUNCT (Hex)
Jump	j	J	PC=JumpAddr	(5) 2_{hex}

J	opcode	address
	31 26	25 0

We first try to find the address of where we want to jump

= 0x0040002C

= 0000 0000 0100 0000 0000 0000 0010 1100

```
Answer:
000010 00000100000000000000001011
0000 1000 0001 0000 0000 0000 0000 1011
0    8    1    0    0    0    0    B
```

As everything is word-aligned, we remove the last 2 bits;

To make it fit, we need to also throw away the first 4 bits.

Q2: MIPS Instruction Encoding

b) Give a simple mathematic expression for the relationship between $s1 and $t0

```
      addi $s1, $zero, 0
loop: srl  $t0, $t0, 1
      beq  $t0, $zero, exit
      addi $s1, $s1, 1
      j    loop
exit:
```

$$\$s1 = \lfloor \log_2(\$t0) \rfloor$$

Break

Take attendance :)

Q3(a)

Binary search

Variable mappings:
addr. of array[0] → $s0;
target → $s1;
lo → $s2;
hi → $s3;
mid → $s4;
ans → $s5.

MIPS code	Comment
`loop:    slt  $t9, $s3, $s2` `         bne  $t9, $zero, end`	# while (lo <= hi) {
`         add  $s4, $s2, $s3` `      [                           ]`	# mid = (lo + hi)/2

`         sll  $t0, $s4, 2`
`         add  $t0, $s0, $t0`
`      [                           ]`
	# t0 = mid*4
	# t0 = &array[mid] in bytes
	# t1 = array[mid]

`         slt  $t9, $s1, $t1` `         beq  $t9, $zero, bigger`	# if (target < array[mid])
`         addi $s3, $s4, -1` `         j    lpEnd`	# hi = mid – 1
`bigger: [                          ]` `        [                          ]`	# else if (target > array[mid])
`         addi $s2, $s4, 1` `         j    lpEnd;`	# lo = mid + 1
`equal:   add  $s5, $s4, $zero` `        [                          ]`	# else { ans = mid; break; }
`lpEnd:  [                          ]`	#} // end of while loop
`end:`	

Q3(a)

Binary search

Variable mappings:
addr. of array[0] → $s0;
target → $s1;
lo → $s2;
hi → $s3;
mid → $s4;
ans → $s5.

MIPS code	Comment
`loop:      slt  $t9, $s3, $s2` `           bne  $t9, $zero, end`	# while (lo <= hi) {
`           add  $s4, $s2, $s3` `        [  srl  $s4, $s4, 1          ]`	# mid = (lo + hi)/2

`           sll  $t0, $s4, 2`
`           add  $t0, $s0, $t0`
`        [  lw   $t1, 0($t0)          ]`
	# t0 = mid*4
	# t0 = &array[mid] in bytes
	# t1 = array[mid]

`           slt  $t9, $s1, $t1` `           beq  $t9, $zero, bigger`	# if (target < array[mid])
`           addi $s3, $s4, -1` `           j    lpEnd`	# hi = mid – 1
`bigger:[   slt  $t9, $t1, $s1    ]` `        [   beq  $t9, $zero, equal ]`	# else if (target > array[mid])
`           addi $s2, $s4, 1` `           j    lpEnd;`	# lo = mid + 1
`equal:     add  $s5, $s4, $zero` `        [  j    end                 ]`	# else { ans = mid; break; }
`lpEnd: [   j    loop               ]`	#} // end of while loop
`end:`	

Q3.

MIPS code			Address
loop:	slt	$t9, $s3, $s2	0xFFFFFF00
	bne	$t9, $zero, end	
	add	$s4, $s2, $s3	
	srl	$s4, $s4, 1	
	sll	$t0, $s4, 2	
	add	$t0, $s0, $t0	
	lw	$t1, 0($t0)	
	slt	$t9, $s1, $t1	
	beq	$t9, $zero, bigger	
	addi	$s3, $s4, -1	
	j	lpEnd	
bigger:	slt	$t9, $t1, $s1	
	beq	$t9, $zero, equal	
	addi	$s2, $s4, 1	
	j	lpEnd;	
equal:	add	$s5, $s4, $zero	
	j	end	
lpEnd:	j	loop	
end:			

(b) What is the immediate value in decimal for the "bne $t9, $zero, end" instruction?

end: is 16 instructions away from bne's next instruction (add $s4, $2, $s3), so the immediate value is **16**.

(c) If the first instruction is at address 0xFFFFFF00, what is the hexadecimal representation of this "j lpEnd"?

Address at lpEnd: is 0xFFFFFF00 + (17_{10}×4) =0xFFFFFF44.
Removing the first 4 bits and last 2 bits, we put this into the immediate field. Opcode of j is 000010. Hence,
000010 1111 1111 1111 1111 1111 0100 01
= **0x0BFFFFD1**

(d) Is the encoding of the second "j lpEnd" different from part (c)?

Same encoding. The two j instructions jump to the same address.

End of Tutorial 3

- 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)

(Also reminder for me to take attendance)

Bonus: Where to go for C

- Look at your standard header files
  - There are more than just stdio.h
- struct and typedef
- Making your own data structures!
  - Really. No linked list, no dictionaries; you need it you make it
- Dynamic memory allocation / malloc + free
  - Valgrind to check for memory leaks

Bonus: MySoC stuffs

- https://mysoc.nus.edu.sg/~newacct to make a new account
- Access to Compute Cluster, SoC email, UNIX servers
- Free printing quota per month (50 pages + 50 pages overdraft)