COMP3211/COMP9211 COMPUTER ARCHITECTURE

Lecturer: Hui Annie Guo
h.guo@unsw.edu.au
K17-501F

Lecture overview

- **Topics**
  - **Course overview**
  - **Introduction to ISA**
- **Suggested reading**
  - **Course outline**
    - **Available on course website**
  - **H&P Chapter 2**

Course overview

- **What is the course about?**
- **Aims of the course**
- **Course organization**
- **Assessments**
- **Other information**

What is this course about?

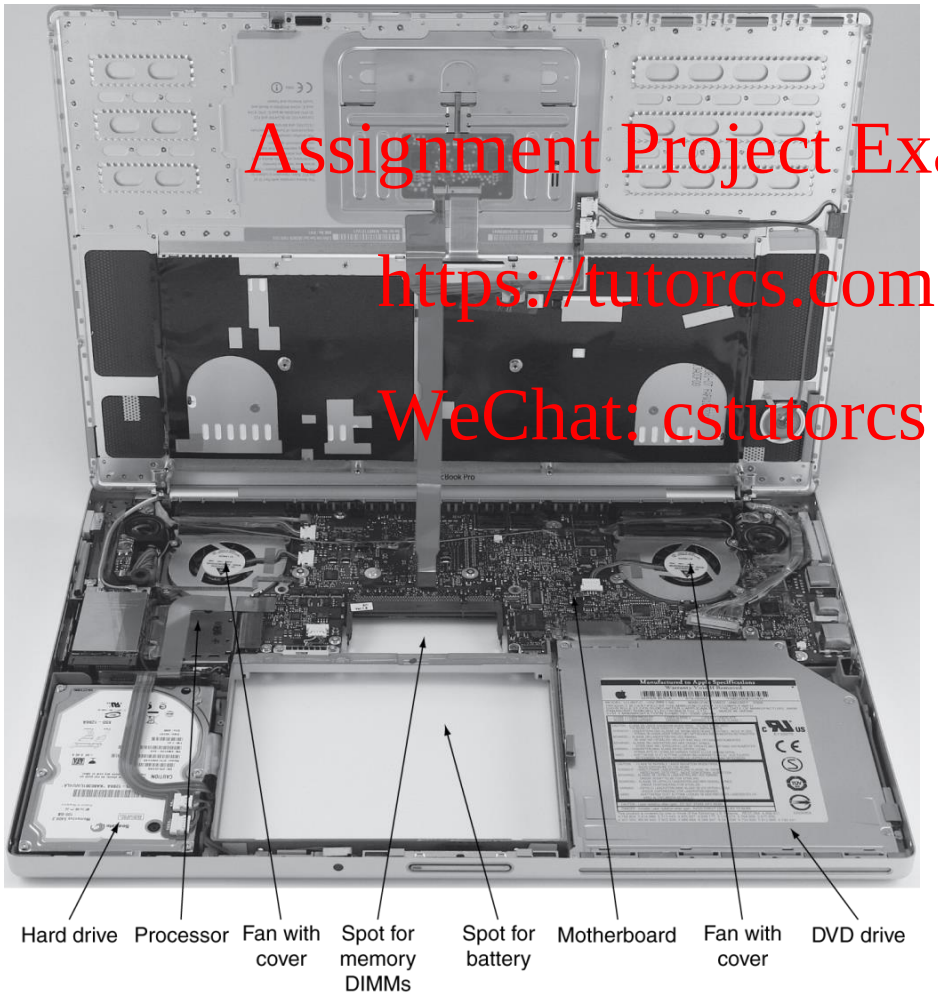
- **This course is about:**
  - **How a computer works**
  - **How a computer is designed**

What is the course about? (cont.)

- **Computer system: HW+SW**
  - **Application software**
    - **For end users**
    - **Often written in high-level language (HLL)**
      - For productivity and portability
  - **System software**
    - **Compiler:** translates HLL programs to machine code
    - **Operating System:** provides services
      - Handling input/output
      - Managing memory and storage
      - Scheduling tasks & resources
  - **Hardware**
    - **Programmable digital system**
      - Processor, memory, I/O controllers

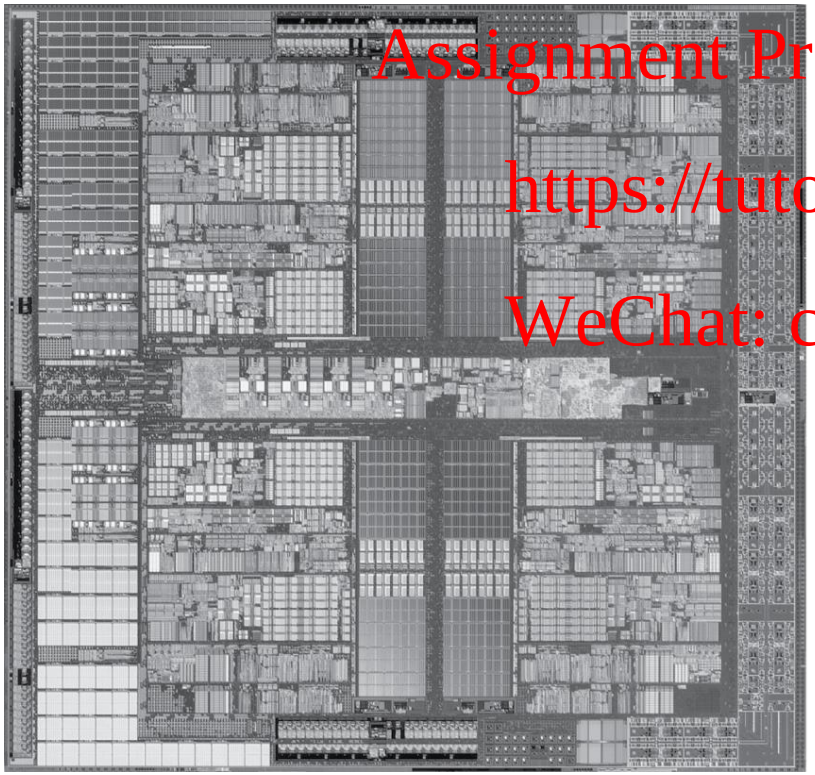
What is the course about? (cont.)

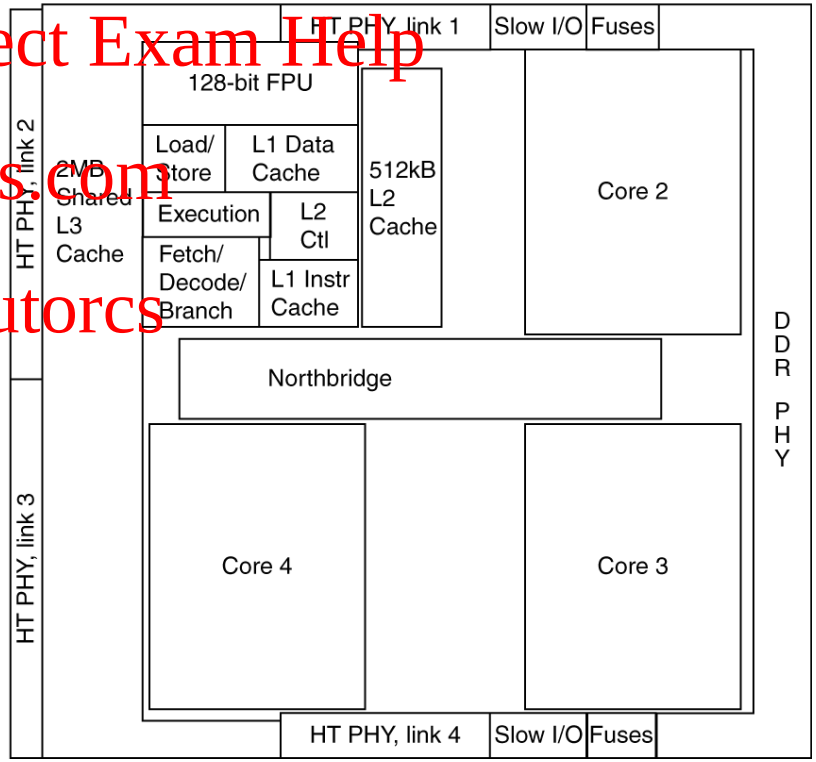
- **Hardware**



Inside processor

- **e.g. AMD Barcelona: 4 processor cores**





What is computer architecture?

- **Frederick P. Brooks:**
  - **Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.**

Turing Awardee:
Frederick Phillips Brooks

What is computer architecture? (cont.)

- **Computer architecture typically includes**
  - **ISA (instruction set architecture)**
  - **Machine organization**

ISA

- **Is the interface between HW and SW of a computer system. It defines attributes/functions**
  - **needed by the software programmer and**
  - **to be implemented by hardware**

Machine organization

- **Is about how functions/features are implemented.**

ISA & organization

- **A family of machines can often share a basic ISA**
  - **To provide code compatibility**
    - **E.g. Intel x86 family**
- **However, organization varies significantly between machine versions**
- **Modern instruction set architectures**
  - **CISC** (Complex Instruction Set Computer)
    - **e.g. VAX, PDP-11, x86, 6800**
  - **RISC** (Reduced Instruction Set Computer)
    - **e.g. SPARC, PowerPC, RISC-V, AVR, ARM, MIPS**

Course outcomes

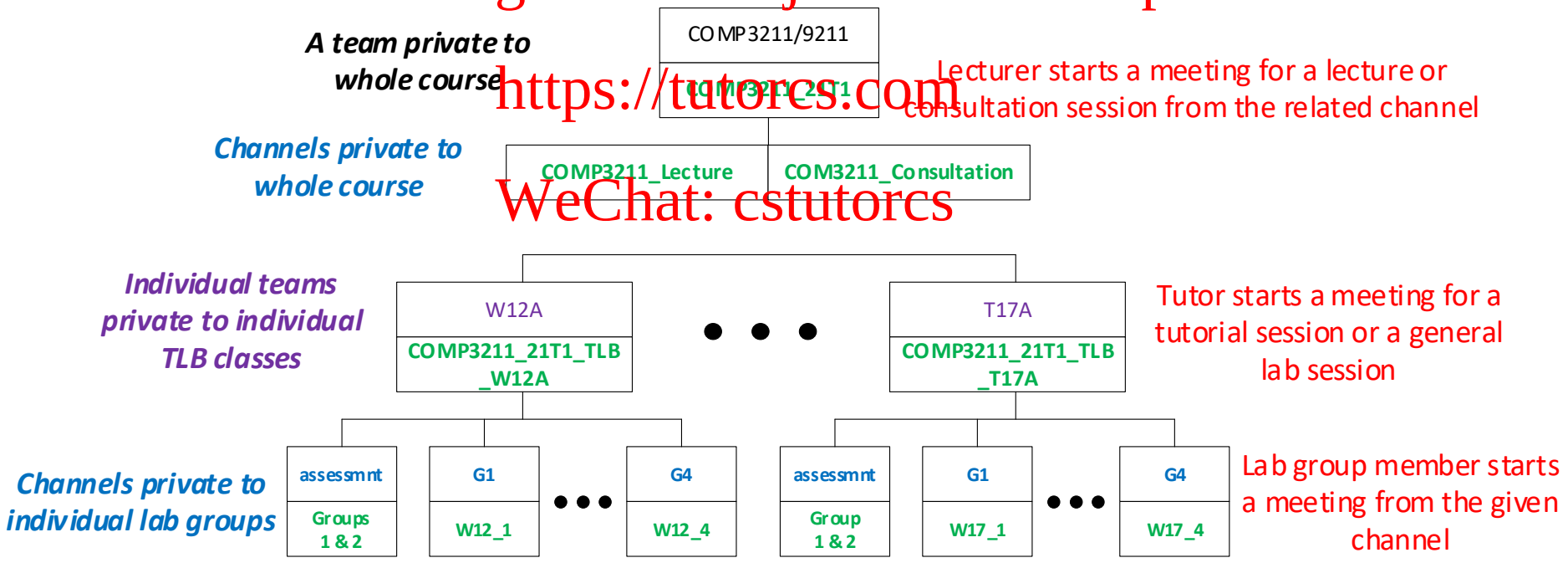
- **By completing the course, you will be**
  - **Able to explain**
    - **How hardware makes the software execution possible,**
    - **How the performance of hardware computer system is evaluated,**
    - **How the computer architecture affects the overall computer system performance**
  - **Competent to apply typical design approaches and techniques in designing a simple RISC processor for a given application**
  - **Capable of describing your design with a hardware description language and evaluate the design with a simulation tool.**

How is the course organized?

- **Lectures**
  - **Two 2-hour lectures each week**
    - **Weeks 1-5, Weeks 7-10 (excl. public holiday)**
  - **Topics**
    - **ISA design**
    - **Processor**
    - **Memory hierarchy**
    - **Parallel processing hardware**
- **Tutorials/Labs (TLB)**
  - **One-hour tutorial and 2-hour lab**
- **Lectures and TLB classes are run in MS Teams**
  - **The general team structure is given in next slide**

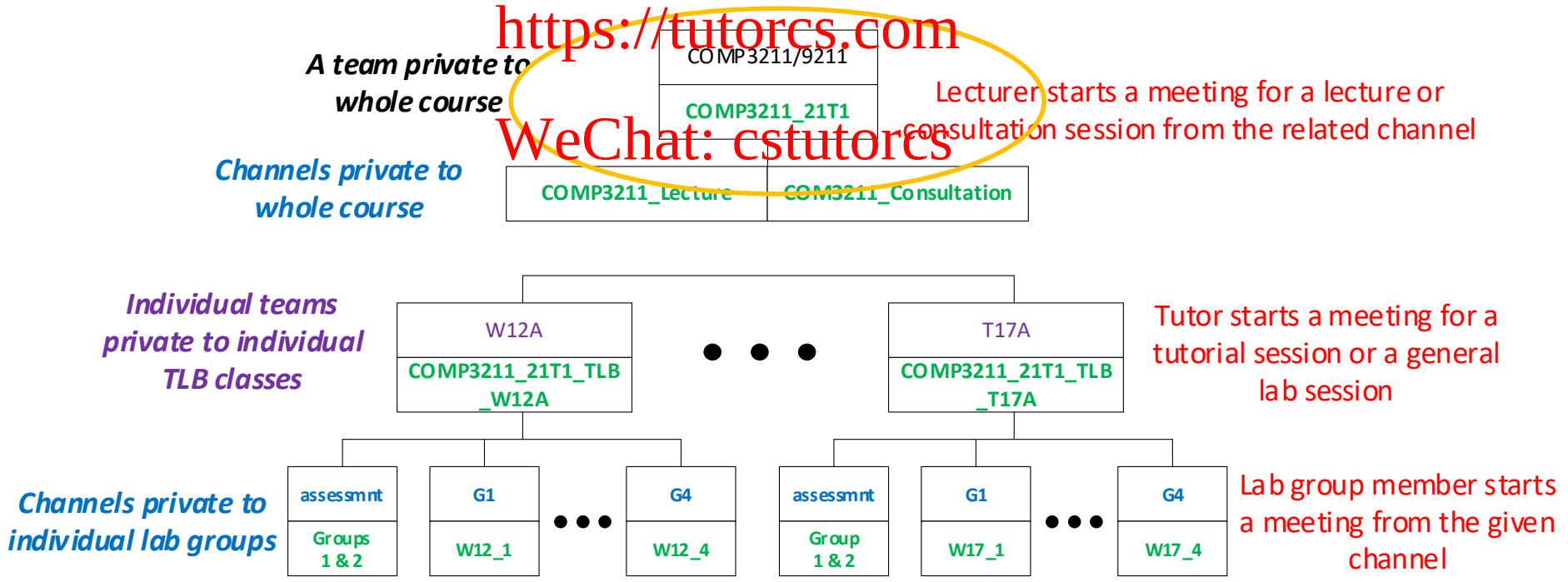
How is the course run with MS Teams?

- **General team structure**
  - **Total five teams**



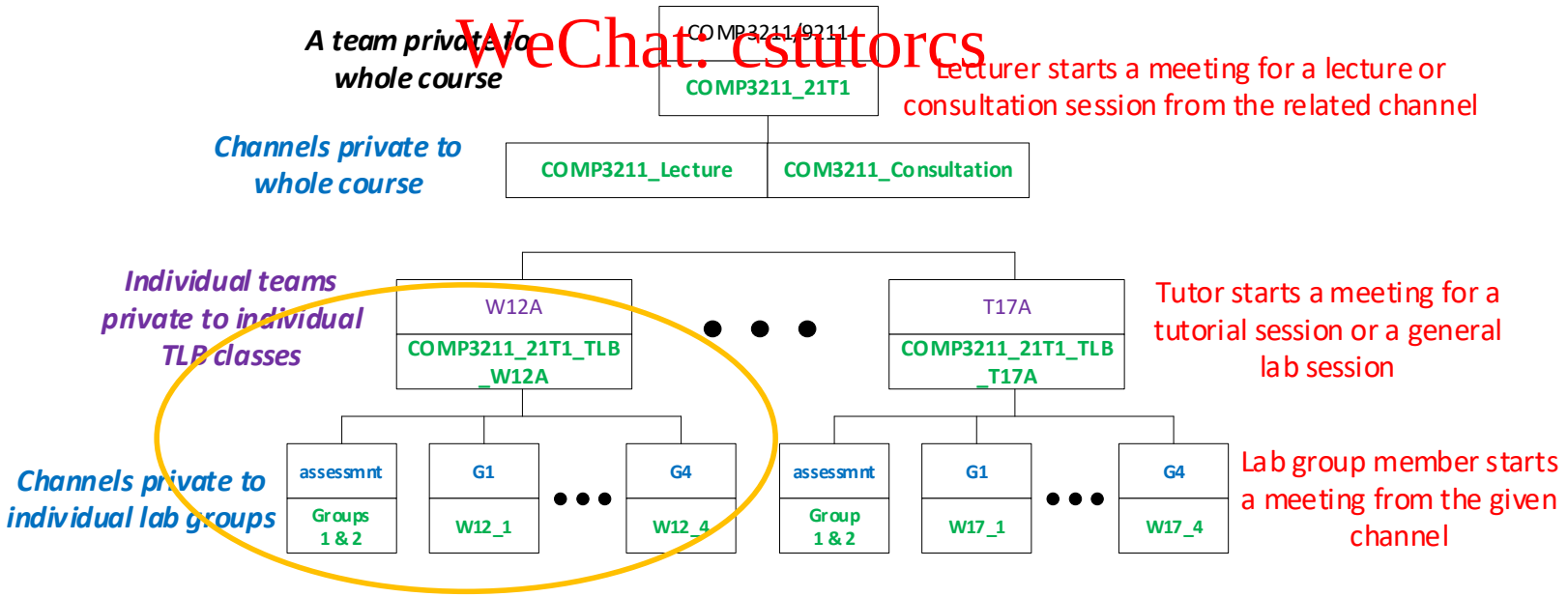
How is the course run with MS Teams? (cont.)

- **Team COMP3211_21T1, consisting of two channels**
  - **For lecture, consultation and general administration**
  - **Each meeting will be started by the lecturer**
    - Join the meeting when you see the meeting-started icon



How is the course run with MS Teams? (cont.)

- **Team COMP3211_21T1_TLB_xxx, consisting of**
  - **Individual channels for each lab group**
    - To be created by the lab tutor after all lab groups are formed in Week 1
    - There are up to 4 channels
  - **Two assessment groups**
    - For peel assessment
  - **See an example in the next slide**



Form a lab group

- **In your lab class this week (Week 1), find your group members**
  - **Typical four members per group**
    - **If required, five-members/group is permitted.**
- **Add your group to the excel file provided in the file folder of your team.**
  - **Instructions are also available in the lab spec of this week.**

Peel assessment

- **For each lab, a TLB class is randomly divided into two groups**
  - **All members in a group assess each other's work.**
  - **An assessment form will be provided**

Tutorials

- **Questions will be released after the second lecture each week**
  - **Mainly in the form of quizzes**
    - **Available in Moodle, a link is provided on the course website**
  - **The solutions will be discussed in the tutorial class of the following week**
- **Assessment distribution**
  - **Quizzes (50%)**
  - **Tutorial class participation (50%)**

Labs

- Three labs
- Xilinx Vivado
  - For modelling and simulation of hardware designs
- Assessment distribution
  - Lab work marks given by your peers (90%)
  - Contribution to the assessment (10%)
    - Add-on benefit: learn from each other

Assignment

- **Application specific processor design**
  - **Done in lab groups**
- **Assessment distribution**
  - **lab demonstration (50%)**
    - **Assessed by tutor**
  - **group presentation (30%)**
    - **Assessed by all course members**
  - **Report (20%)**
    - **Assessed by lecturer**

Overall assessment distribution

- **Assignment**
  - **20%**
- **Lab**
  - **20%**
- **Tutorial**
  - **10%**

- **For the group work, there may be some adjustments if members in a group have very unbalanced contributions**

Overall assessment distribution (cont.)

- **Final exam (online)**
  - **2 hours**
  - **50%**
- **To pass the course, you must have**
  - **final result >= 50 (out of 100), and**
  - **final exam >= 40 (out of 100)**

Staff

- **Annie Guo**
  - **h.guo@unsw.edu.au**
  - **Consultation: 3-5pm Fri.**
    - **For face-to-face consultation, please make an appointment.**
- **Brian Udugama**
  - **b.udugama@unsw.edu.au**
- **Kenny Dow**
  - **h.dow@unsw.edu.au**

Textbook and references

- **Textbook**
  - **Computer Organization and Design: The Hardware/Software Interface, D.A. Patterson and J.L. Hennessy, 5th Ed., Morgan Kaufmann, 2014**
    - **Print:** https://www.bookshop.unsw.edu.au/details.cgi?ITEMNO=9780124077263
    - **Digital**: https://unswbookshop.vitalsource.com/products/-v9780124078864
- **References and software tool**
  - **See the course website for a list of references and downloadable software**
- **Lecture notes**
  - **Posted each week before lecture**
- **Lecture recordings**
  - **Available after each lecture**

Support

- **Course website**
  - **http://www.cse.unsw.edu.au/~cs3211**
  - **Regularly check the Notices page for the course administration announcements.**

- **Course staff**

  - **The staff contact information can also be found on course website**

INTRODUCTION TO INSTRUCTION SET ARCHITECTURE AND DESIGN

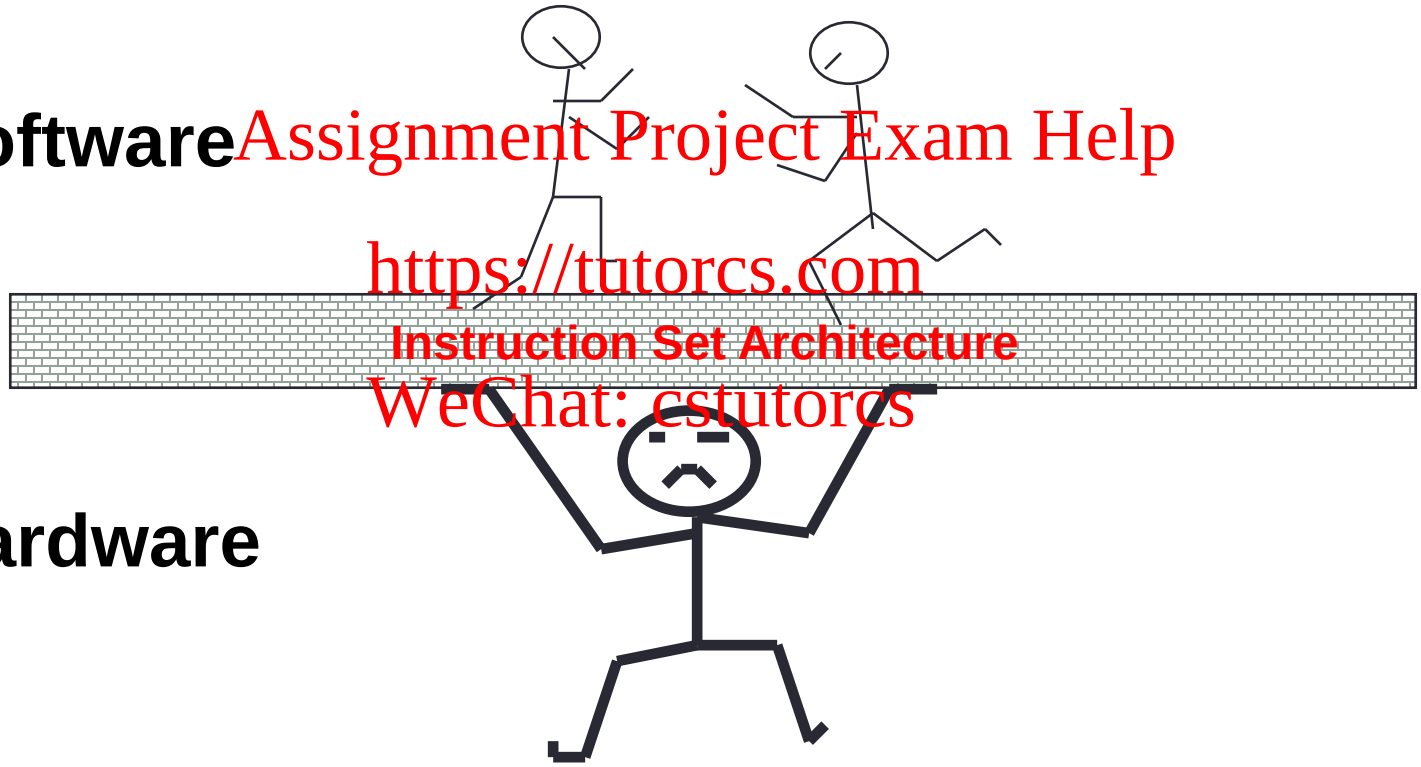
Lecturer: Hui Annie Guo
h.guo@unsw.edu.au
K17-501F

Lecture overview

- What is instruction set architecture?
- Four design principles
- Typical design issues and guidelines
- MIPS instruction set

ISA - interface between HW/SW

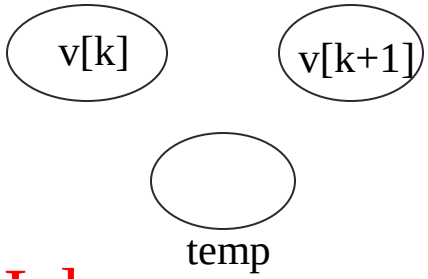
software



hardware

Levels of representation

v[k] ←→ v[k+1]



High Level Language Program

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program

```
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
```

Assembler

Machine Language Program

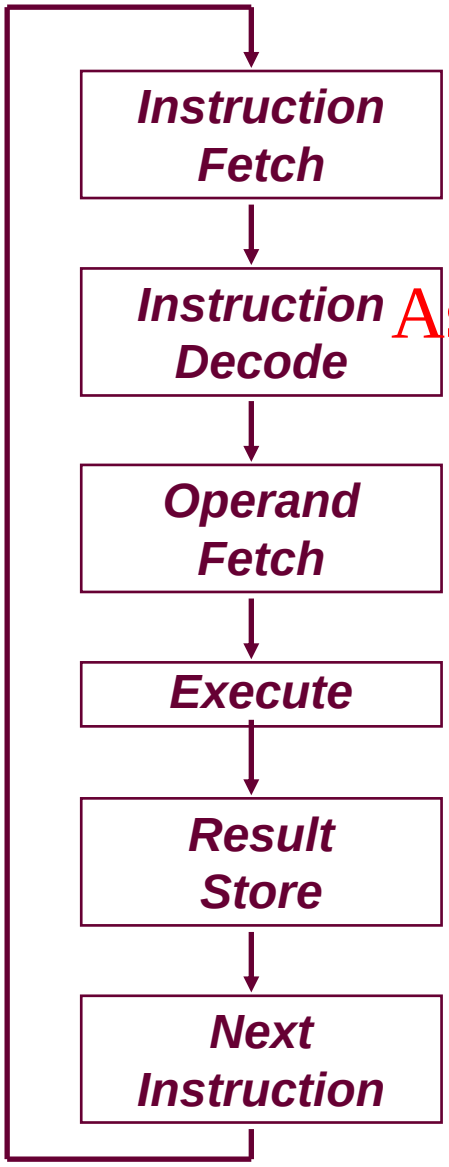
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Machine Interpretation

Control Signal Specification

Execution cycle

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



Obtain instruction from storage

Determine required actions

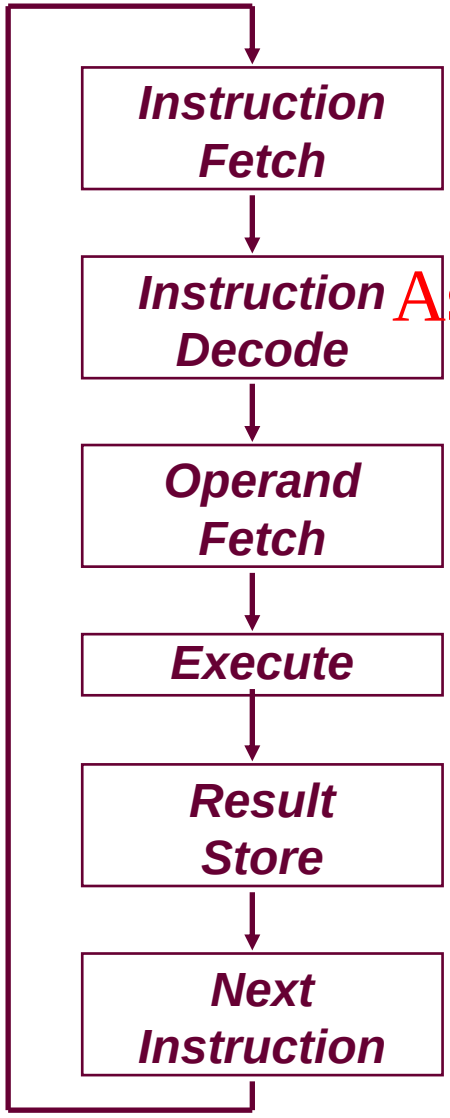
Locate and obtain operands

Compute result

Save result in storage for later use

Determine next instruction to be executed

ISA: what are needed?



- **Operations**
  - **What basic operations are supported?**
- **Data type and size**
- **Operands**
  - **How many and where to find?**
- **Instruction format and encoding**
  - **How to represent an instruction?**
- **Next instruction**
  - **Branch? Any condition?**

Instruction set architecture design

- **Goal:**
  - **The instruction set should be easy to implement, good for performance, and possibly more**

- **Four design principles:**

  - **Smaller is faster**

  - **Simplicity favors regularity**
  - **Make the common case fast**
  - **Good design demands a compromise**

MIPS example of four design principles

- **Simple** (simplicity favors regularity)
  - **MIPS instructions are all 32 bits in size**
  - **Arithmetic instructions always have three operands**
  - **Arithmetic operations are performed on registers**
- **Small** (smaller is faster)
  - **MIPS has a small register file of only 32 registers, each with 32 bits**
- **Compromise** (good design demands a compromise)
  - **MIPS has three instruction formats**
- **Optimizing common case** (making a common occurrence fast)
  - **Immediate values are provided in I-type instructions**

General purpose registers

- **Like memory and stack, general purpose registers can hold variables.**
  - **Compared to memory, registers help**
    - **Improve performance**
      - Register is faster than memory
    - **Reduce code size**
      - since register takes fewer bits than memory location
  - **Compared to stack, registers are easy to use**
    - **e.g., (A*B) – (C*D) – (E*F), multiplications can be done in any order, whereas with the stack they can't.**

MIPS integer registers

- R0-R31 or $0-$31
- They are also divided into groups for special uses.

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		
15	t7	
16	s0	callee saves
. . .		
23	s7	
24	t8	temporary
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	Frame pointer
31	ra	Return Address (HW)

Memory addressing

- Most machines use byte address
- Two issues of multi-byte objects stored in memory:
  - Endianness
    - The order of the multiple bytes stored in the memory
  - Alignment
    - The boundary of the multi-byte object in the memory

Endianness and alignment

- **For a word of multi-bytes**
  - **Big Endian:**
    - **address of most significant byte = word address**
      - Most significant byte stored first
    - **IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA**
  - **Little Endian:**
    - **address of least significant byte = word address**
      - Least significant byte stored first
    - **Intel 80x86, DEC Vax, DEC Alpha (Windows NT)**
  - **Alignment:**
    - **words fall on addresses that are multiple of their size.**

Possible addressing modes

Addressing mode	Example	Meaning
Immediate	Add R4,#3	R4 ← R4+3
Register	Add R4,R3	R4← R4+R3
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Indexed / Base	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,–(R2)	R2 ← R2–d; R1 ← R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1 ← R1+Mem[100+R2+R3*d]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]

Which modes to use?

- It is too expensive to implement all of possible addressing modes
- How to decide which to use?
  - Via analysis and profiling
  - See example next

Example:

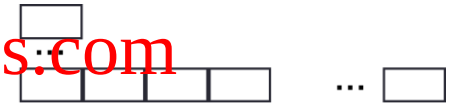
- **Three programs measured on the VAX machine that has all addressing modes implemented**
  - **Addressing mode usage**

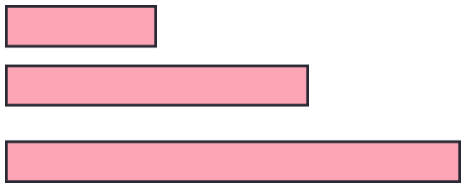
displacement:	42% avg, 32% to 55%
immediate:	33% avg, 17% to 43%
register indirect:	13% avg, 3% to 24%
scaled:	7% avg, 4% to 16%
memory indirect:	3% avg, 1% to 6%
misc:	2% avg, 0% to 3%

    - **displacement & immediate: 75%**
    - **displacement & immediate & register indirect: 88%**
  - **Constant value sizes**
    - **For displacement, 12-16 bits: 99%**
    - **For immediate 8-16 bits: , 99%**

Instruction format

- **Instruction formats can be categorized based on the instruction width.**
  - **Varied**
    - **Each instruction uses its own required width**
  - **Fixed**
    - **All instructions have the same width**
  - **Hybrid**
    - **Instructions are divided into several groups**
    - **Instructions in each group have a fixed width**





Instruction format: some design strategies

- If code size is most important
  - use variable width instructions
    - as in some embedded apps
- If performance is most important,
  - use fixed width instructions
    - E.g. MIPS
- Embedded machines added optional mode to execute subset of wide instructions (Thumb, MIPS16)
  - To trade between performance and density

MIPS instruction set architecture

- **We'll be working with the MIPS instruction set architecture**
  - **A typical RISC ISA**
  - **Used by many computer system designers**
    - **ATI Technologies, Broadcom, NEC, Nintendo, Cisco, Silicon Graphics, Sony, …**

Features of MIPS ISA

- All instructions are of 32 bits
  - in 3 formats
- Arithmetic and logic operations are always performed on registers
  - reg-reg AL instructions
- Having 32 x 32-bit integer registers and 32 FP registers
- Single address mode for accessing data in memory
  - base + displacement
- Simple branch conditions
  - compare two registers for equal/not equal

MIPS 3 instruction formats

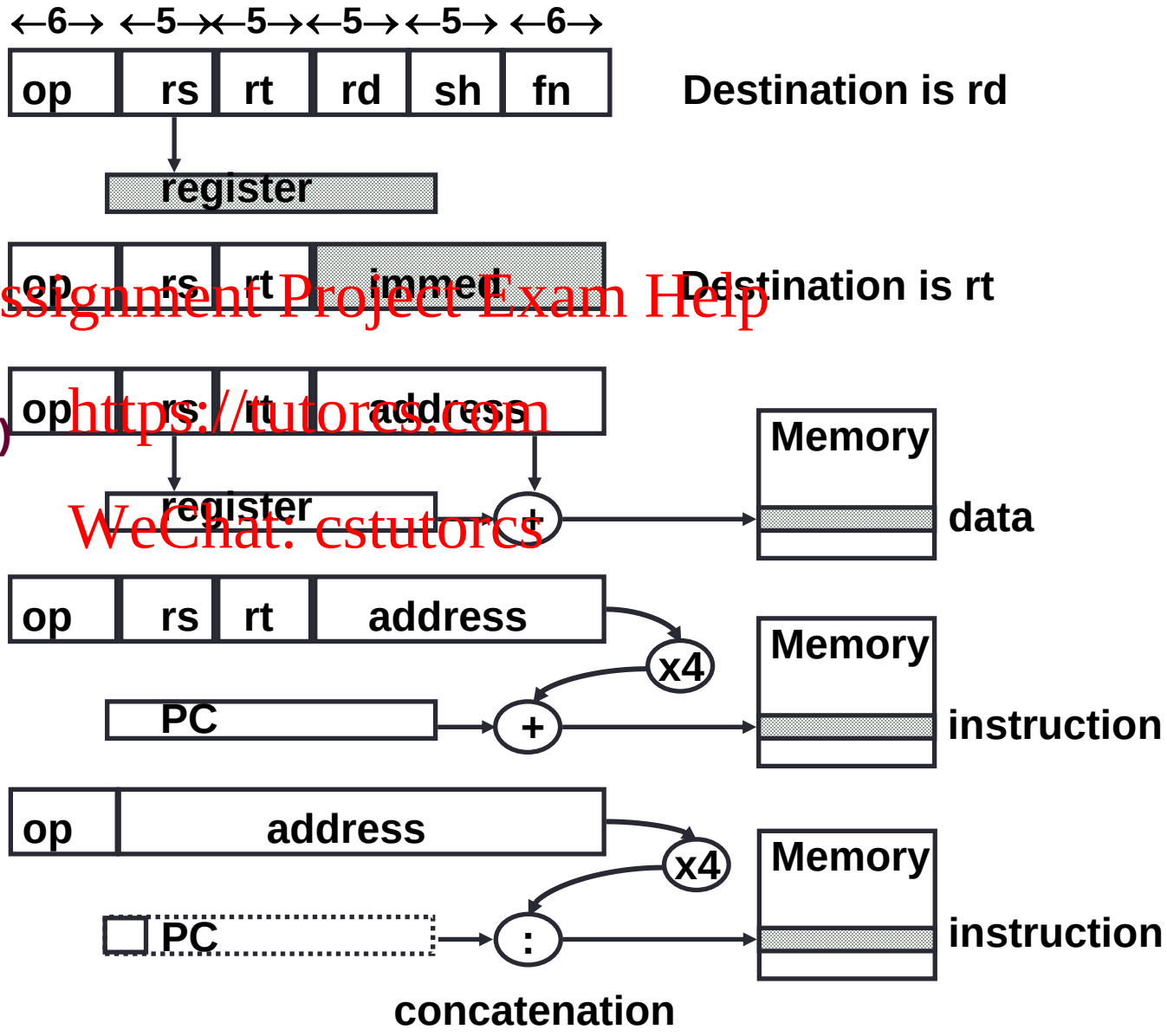
R-type

I-type

J-type

MIPS 5 addressing modes / 3 instruction formats

1. Register (direct) (e.g. addition)
2. Immediate (e.g. increment)
3. Base+index (e.g. array access)
4. PC-relative (e.g. branch)
5. Pseudodirect (e.g. jump)



MIPS arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands;
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands;
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant;
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands;
add imm. unsign.	addiu $1,$2,10	$1 = $2 + 10	+ constant;
subtract unsign.	subu $1,$2,$3	$1 = $2 – $3	3 operands;
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsign.	multu $2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
divide unsign.	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient & remainder
move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

MIPS logical & shift instructions

Instruction	*Example*	*Meaning*	*Comment*
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands; Logical AND
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands; Logical OR
xor	xor $1,$2,$3	$1 = $2 ⊕ $3	3 reg. operands; Logical XOR
nor	nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. operands; Logical NOR
and immediate	andi $1,$2,10	$1 = $2 & 10	Logical AND reg, constant
or immediate	ori $1,$2,10	$1 = $2 | 10	Logical OR reg, constant
xor immediate	xori $1, $2,10	$1 = ~$2 &~10	Logical XOR reg, constant
shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
shift right arithm.	sra $1,$2,10	$1 = $2 >> 10	Shift right (sign extend)
shift left logical	sllv $1,$2,$3	$1 = $2 << $3	Shift left by variable
shift right logical	srlv $1,$2, $3	$1 = $2 >> $3	Shift right by variable
shift right arithm.	srav $1,$2, $3	$1 = $2 >> $3	Shift right arith. by variable

MIPS data transfer instructions

Instruction	*Example*	*Meaning*	*Comment*
store word	sw $3, 500($4)	memory[$4+500] ← $3	one word
store half	sh $3, 502($2)	memory[$2+502] ← $3	half word
store byte	sb $2, 41($3)	memory[$3+41] ← $2	one byte
load word	lw $1, 30($2)	$1 ← memory[$2 + 30]	one word
load half	lh $1, 40($3)	$1 ← mem[$3+40]	half word
load half word unsign.	lhu $1, 40($3)	$1 ← mem[$3+40]	half word
load byte	lb $1, 40($3)	$1 ← mem[$3+40]	one byte
load byte unsign.	lbu $1, 40($3)	$1 ← mem[$3+40]	one byte
load upper imm.	lui $1, 40	$1 ← 40<<16	

MIPS jump, branch, compare instructions

Instruction	*Example*	*Meaning*
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100
set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0
set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0
set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0
set l. t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0
jump	j 10000	go to 10000
jump register	jr $31	go to $31
jump and link	jal 10000	$31 = PC + 4; go to 10000

Homework

- Browse the course website
- Skim through textbook
- Read the chapter related MIPS ISA
- Install Xilinx Vivado on your home machine
  - The installation guide and related docs are available on the Resources page on the course website