CSE 140
Computer Architecture

Lecture 1 – General Course Overview

CSE 140: Fall 2019

- Lecturer
  - Chi Yan “Daniel” Leung
  - cleung3@ucmerced.edu
  - Office Room: AOA 126
  - Office Hours:
    - T/R: 10:00am – 12:00pm
    - W: 3:00pm – 4:30pm
    - By appointment
- TA
  - Nasit Sony nsony@ucmerced.edu
  - Rakesh Jasti rjasti@ucmerced.edu
- All email inquiries received before 5pm during school days will be replied within 48 hours
  - Please follow the guidelines below for proper email communications
    - How_to_Email_your_Professor.pdf (at CatCourses)

Course Overview

- CatCourses
  - Check regularly for announcements.
  - Labs, Projects, and Reading/Homework Assignments will be posted and submitted there.
  - Grades for assignments will also be found there (secure).
- 2 Lectures and 1 Lab per week
- 2 Mid-term exams (Oct. 10 and Nov. 21, in class)
- 2 Final exams (Friday, 12/20, in class)
- Regular quizzes
- 10 Lab/HW assignments
- 3 projects

Course Objectives

- Hardware design
  - Study topics learned from CSE 31 in more detail
    - Datapath
    - Single Cycle CPU
    - Pipelining
    - Cache
  - More advanced topics in modern computer hardware
    - Virtual memory
    - Complex pipeline
    - Branch prediction
    - Parallel processing
    - GPU

Course Objectives

- Labs/HW:
  - Giving each other help in finding bugs and in understanding the assignment is perfectly acceptable.
  - Show your work to your TA before leaving the lab for participation credit.
  - No late submission is allowed after 2 days beyond the due time.
  - Try to debug yourself before asking questions
  - Follow the guidelines (see below) to debug and ask your TA for help.
  - http://www.cplusplus.com/forum/articles/28767/
  - http://www.catb.org/esr/faqs/smart-questions.html

Course Material

- Text Books:
  - Computer Organization and Design from zyBooks
    - Sign up/sign in at zyBooks.com
    - Enter zyBook code: ***UCMERCEDCSE140LeungFall2019***
    - **You must subscribe your own copy using your UCM email address.** Participation grade will be partly evaluated based on the activities within the subscription account.

Grading

Labs/HW:	15%
Projects:	20%
Mid-terms:	20%
Final exams (comprehensive):	25%
Quizzes:	10%
Participation	10%

- Grades:
  - 90% of points at least an A
  - 80% at least a B
  - 70% at least a C

Hints for success

- Attend lecture
- Read the textbook
- Do & understand the labs and homework YOURSELF
- Create a portfolio to save all your work
- Take notes while reading and in lecture
- Ask questions

Policies

- Don’t copy someone else’s code
- Don’t give your code away
- Don’t outsource your assignments
- Don’t use electronic devices in exams (includes calculators)
- Don’t use electronic devices during lecture for purposes other than note taking
- Turn off speakers/cellphone during class

No Cheating!

- Communicating information to another student during examination.
- Knowingly allowing another student to copy one’s work.
- Offering another person’s work as one’s own.
- I am serious!
- See new CSE policy at CatCourses

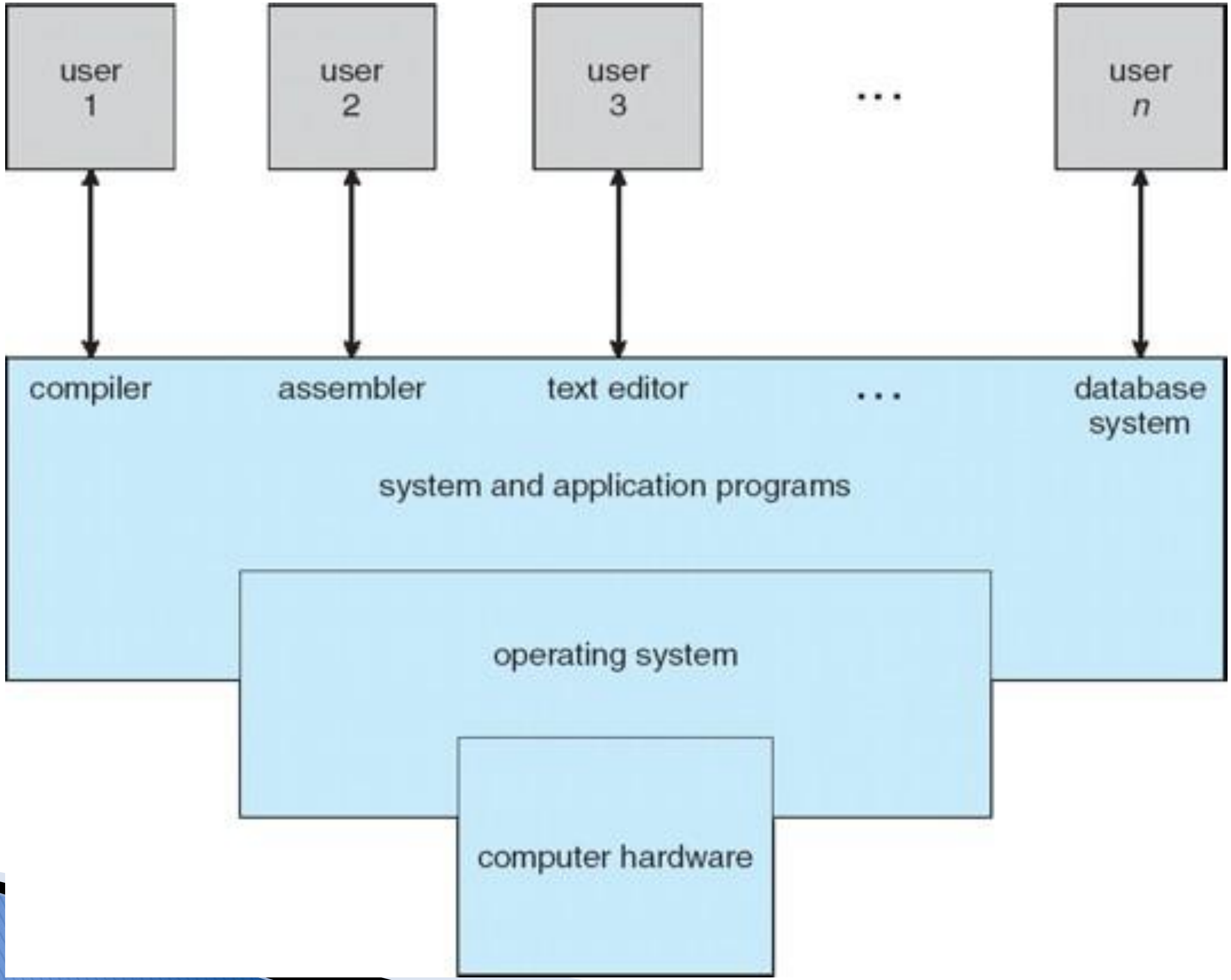
What is a Computer?

- Examples?
- Desktop Computing
  - Emphasis on price-performance
- Personal Mobile Device (PMD)
  - e.g. smart phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for "Software as a Service (SaaS)"
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

What is Computer Architecture?

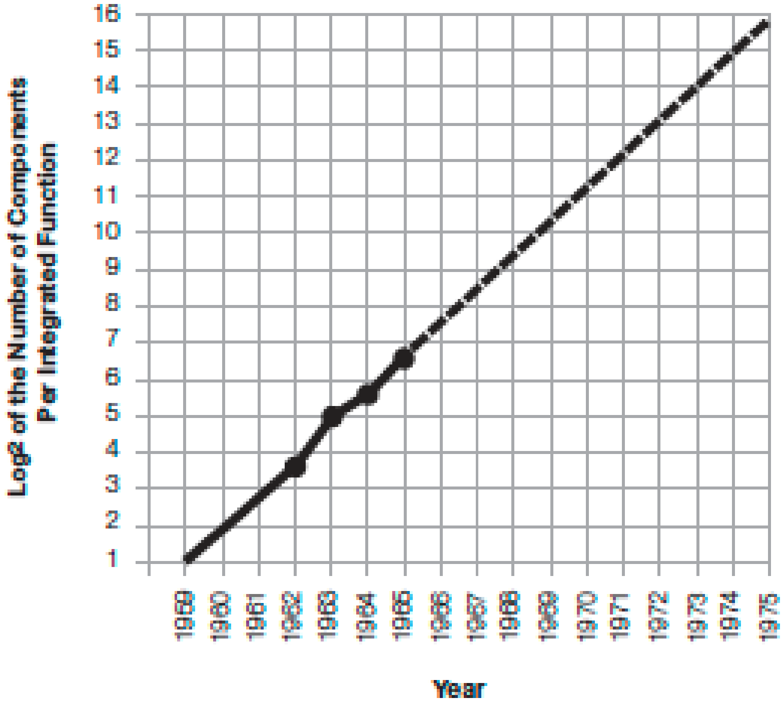
- “Old” view of computer architecture:
  - Instruction Set Architecture (ISA) design
  - Example: CISC vs RISC
    - Complex instruction set computing vs Reduced instruction set computing
  - i.e. decisions regarding:
    - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding
- “Real” computer architecture:
  - Specific requirements of the target machine
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware
- Why should we care?

Computer System Components



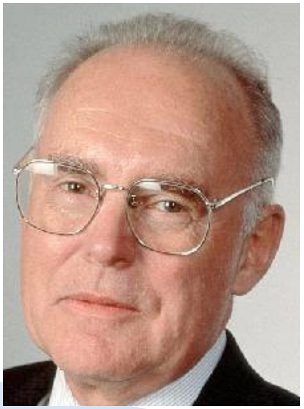
Moore’s Law

“The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. …That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.” *(from 50 in 1965)*



Gordon Moore, “Cramming more components onto integrated circuits,” *Electronics*, Volume 38, Number 8, April 19, 1965

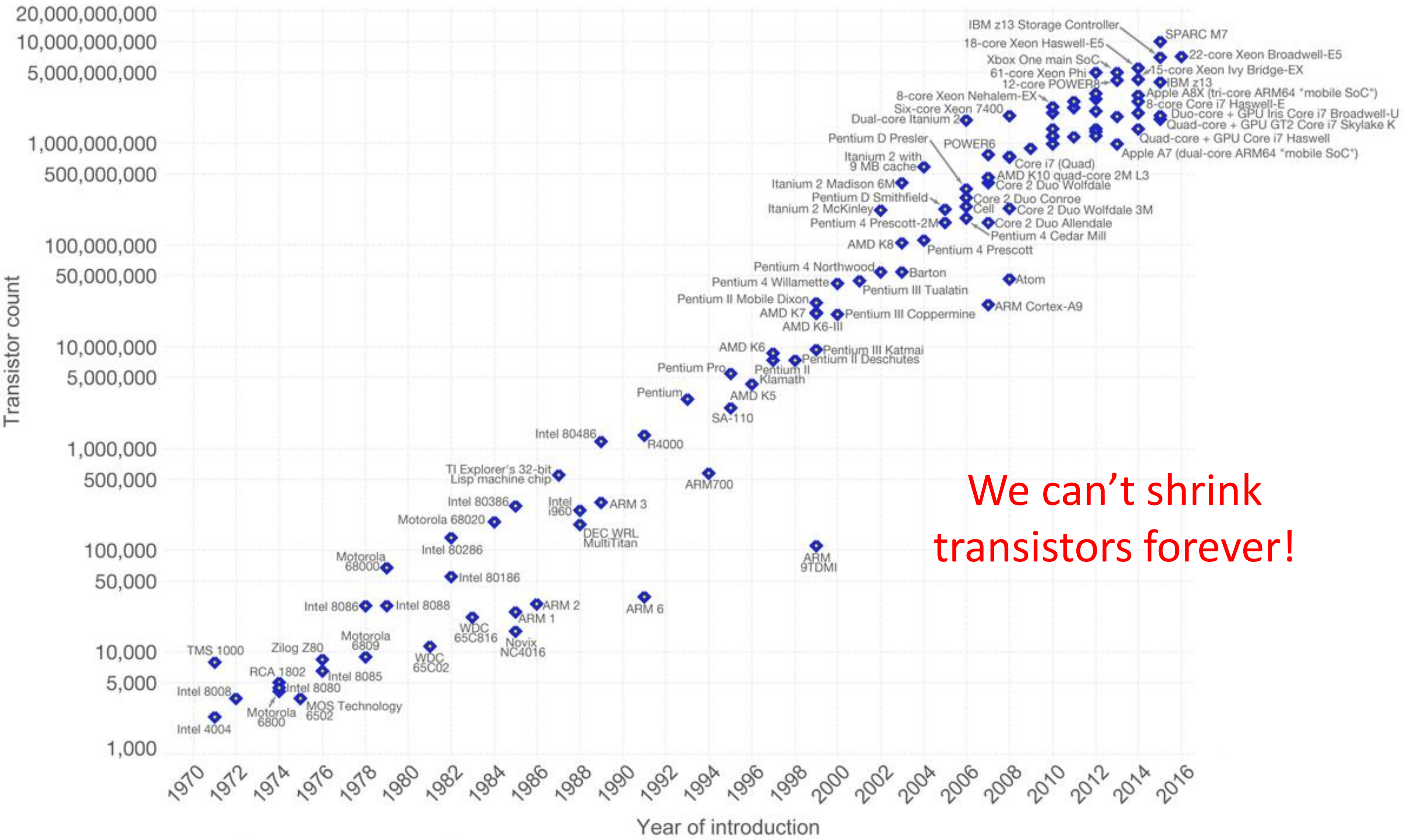
Gordon Moore
Intel Cofounder



Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



We can't shrink transistors forever!

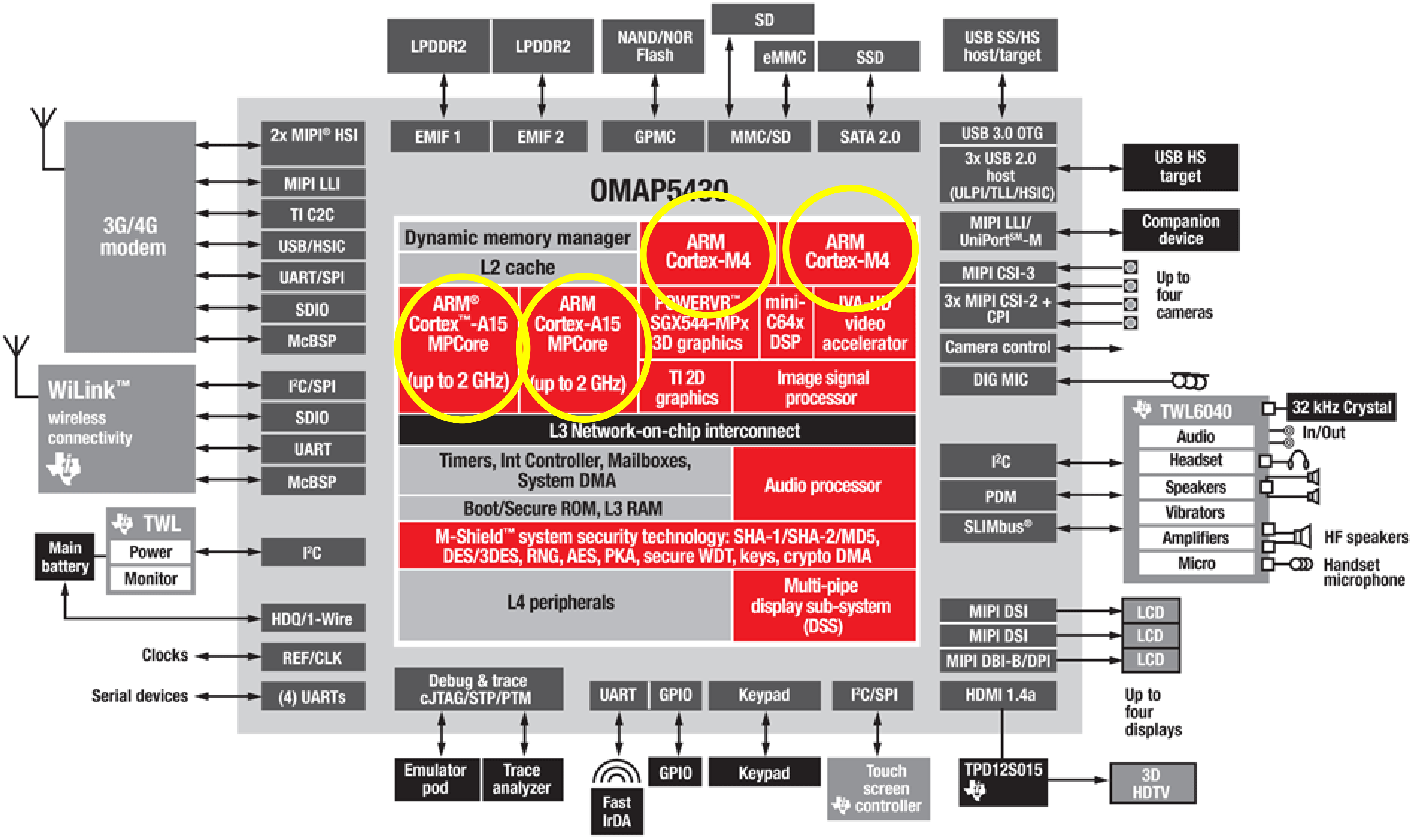
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

TI OMAP5430 SoC (System on a Chip)

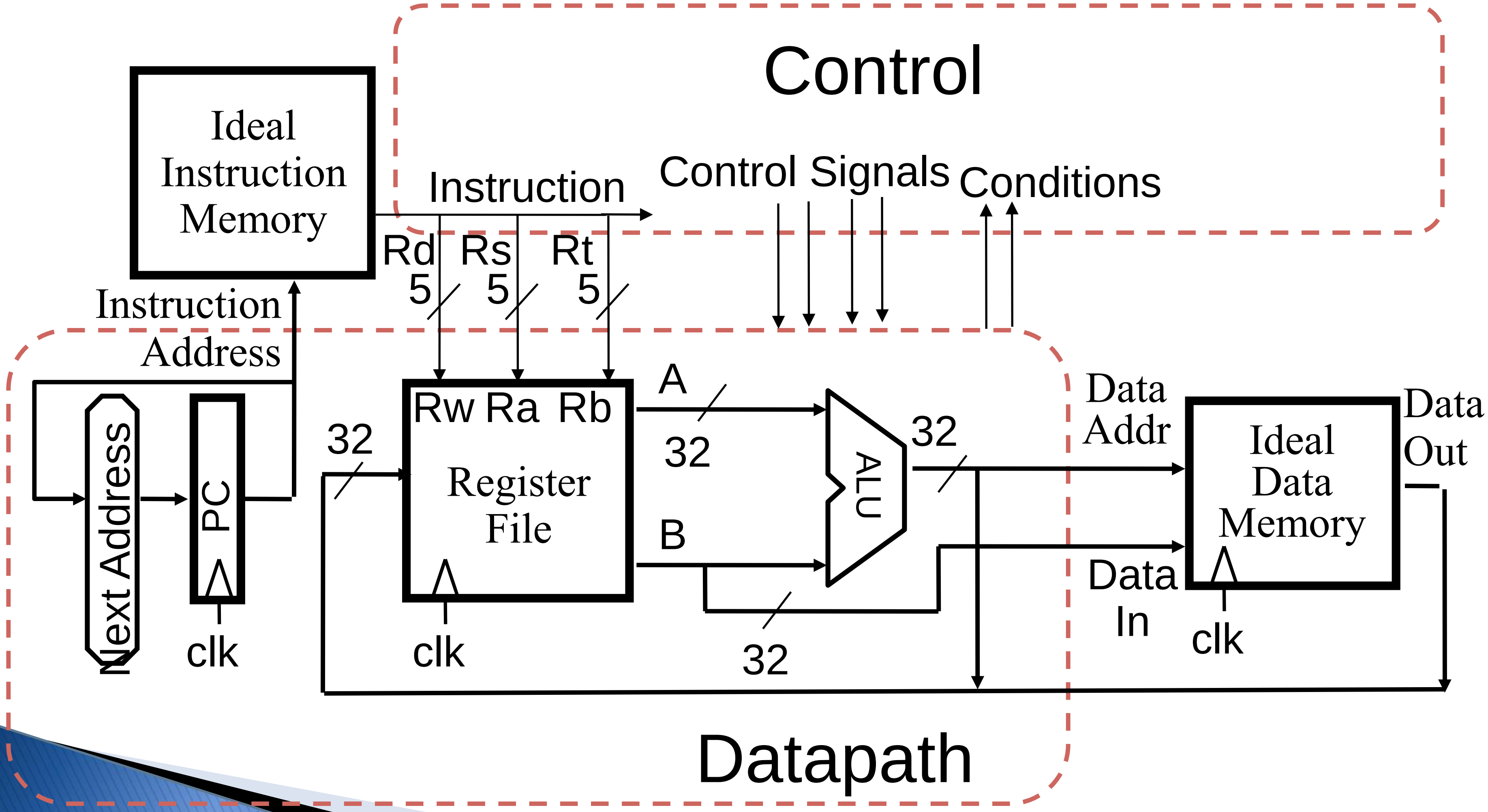
Not a CPU!

Available in 2013

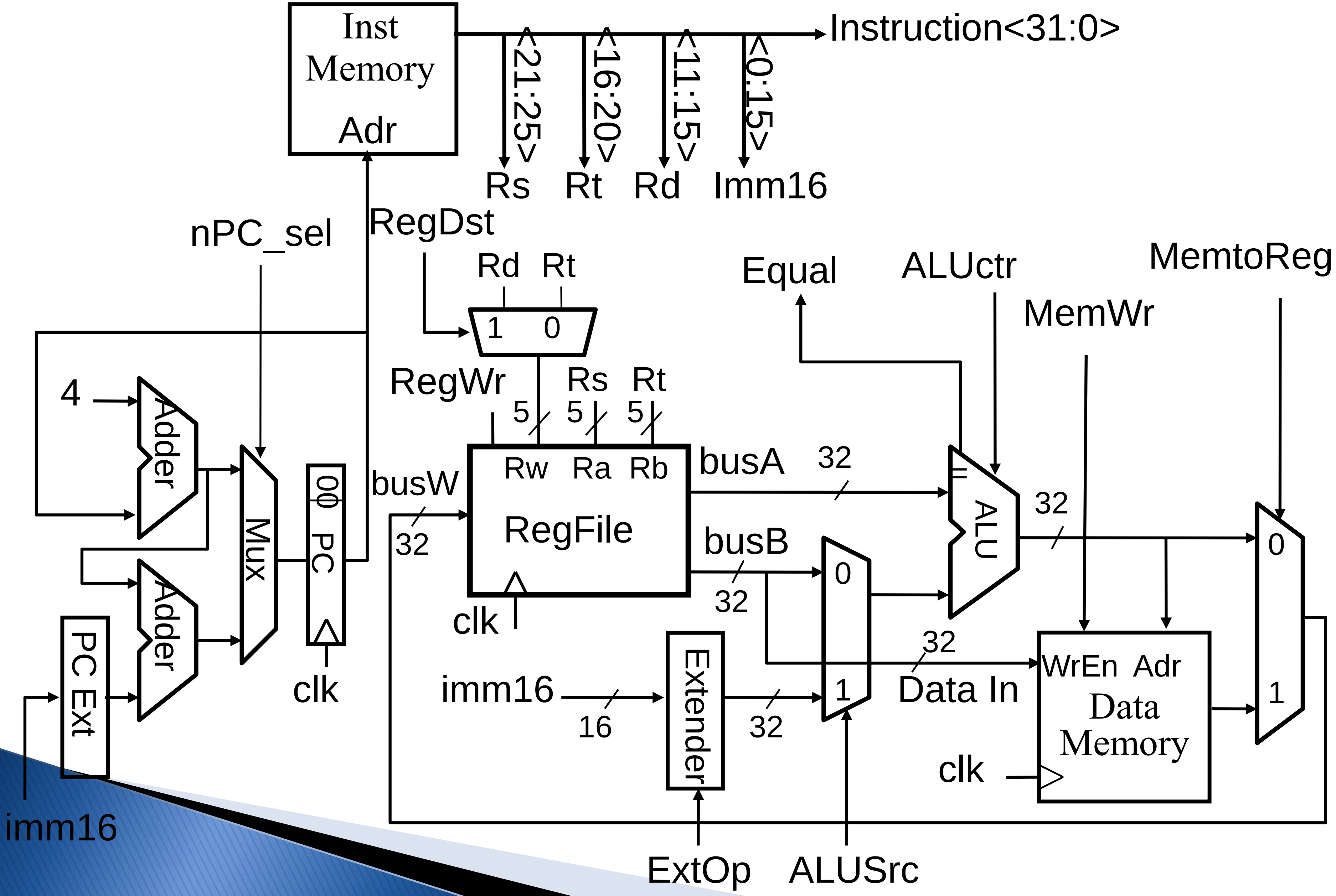
TI OMAP5430 SoC



Abstract View of the CPU Implementation



Datapath


Inst
Memory
Adr
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rs
Rt
Rd
Imm16
nPC_sel
RegDst
Rd
Rt
1
0
Equal
ALUctr
MemtoReg
MemWr
4
Adder
Adder
Mux
00
PC
clk
PC Ext
imm16
RegWr
Rs
Rt
5
5
5
Rw
Ra
Rb
RegFile
busW
32
busA
32
busB
32
clk
=
ALU
32
0
1
0
1
imm16
16
Extender
32
32
Data In
WrEn
Adr
Data
Memory
clk
ExtOp
ALUSrc

MIPS: operation Overview

- Arithmetic Logical:
  - Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU
  - AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, LUI
  - SLL, SRL, SRA, SLLV, SRLV, SRAV
- Memory Access:
  - LB, LBU, LH, LHU, LW, LWL,LWR
  - SB, SH, SW, SWL, SWR

MIPS logical instructions

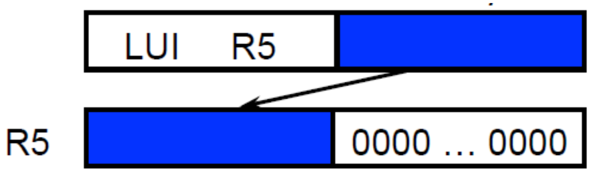
Instruction	*Example*	*Meaning*	*Comment*
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands; Logical AND
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands; Logical OR
xor	xor $1,$2,$3	$1 = $2 ^ $3	3 reg. operands; Logical XOR
nor	nor $1,$2,$3	$1 = ~($2 | $3)	3 reg. operands; Logical NOR
and immediate	andi $1,$2,10	$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 arith	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 arith	srav $1,$2, $3	$1 = $2 >> $3	Shift right arith. by variable

How to multiply or divide by 2^i ?

MIPS: data transfer instructions

Instruction	***Comment***
sw $3, 500($4)	Store word
sh $3, 502($2)	Store half
sb $2, 41($3)	Store byte
lw $1, 30($2)	Load word
lh $1, 40($3)	Load halfword
lhu $1, 40($3)	Load halfword unsigned
lb $1, 40($3)	Load byte
lbu $1, 40($3)	Load byte unsigned
lui $1, 40	Load Upper Immediate (16 bits shifted left by 16)

Q: Why need lui?



MIPS: arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsigned	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 unsigned	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

Q: Which add for address arithmetic?
Which add for integers?

MIPS: Control Instructions

Instruction	***Example***	***Meaning***
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100 Equal test; PC relative branch
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100 Not equal test; PC relative
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 Compare < constant; 2's comp.
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 Compare < constant; natural numbers
jump	j 10000	go to 10000 Jump to target address
jump register	jr $31	go to $31 For switch, procedure return
jump and link	jal 10000	$31 = PC + 4; go to 10000 For procedure call

Quiz

Given: $s3=i, $s4=j, $s5=@A

```
Loop:  addiu $s4,$s4,1      # j = j + 1
       sll   $t1,$s3,2      # $t1 = 4 * i
       addu $t1,$t1,$s5     # $t1 = @ A[i]
       lw    $t0,0($t1)     # $t0 = A[i]
       slti  $t1,$t0,10     # $t1 = $t0 < 10
       beq   $t1,$0, Loop # goto Loop
       addiu $s3,$s3,1      # i = i + 1
       slti  $t1,$t0, 0     # $t1 = $t0 < 0
       bne   $t1,$0, Loop # goto Loop
```

What C code properly fills in the blank in loop?

```
do
    j = j +
1;
while
 (______);
```

1: A[i++] >= 10

2: A[i++] >= 10 || A[i] < 0

3: A[i]   >= 10 || A[i] < 0

4: A[i]   >= 10 || A[i++] < 0

5: A[i]   >= 10   && A[i++] < 0

Quiz

Given: $s3=i, $s4=j, $s5=@A

```
Loop:  addiu $s4,$s4,1      # j = j + 1
       sll    $t1,$s3,2     # $t1 = 4 * i
       addu $t1,$t1,$s5     # $t1 = @ A[i]
       lw     $t0,0($t1)    # $t0 = A[i]
       slti   $t1,$t0,10    # $t1 = $t0 < 10
       beq    $t1,$0, Loop  # goto Loop
       addiu $s3,$s3,1      # i = i + 1
       slti   $t1,$t0, 0    # $t1 = $t0 < 0
       bne    $t1,$0, Loop  # goto Loop
```

What C code properly fills in the blank in loop?

```
do
    j = j +
1;
while
 (_______);
```

1: A[i++] >= 10

2: A[i++] >= 10 || A[i] < 0

3: A[i] >= 10 || A[i] < 0

4: A[i] >= 10 || A[i++] < 0

5: A[i] >= 10 && A[i++] < 0

Announcement

- Lab #1 starts next week
- Reading assignment
  - Chapter 2.12, 4.1 – 4.3
    - Do all Participation Activities in each section
    - Access through CatCourses
    - Due Thursday (9/5) at 11:59pm
  - Review CSE 31 materials (available at CatCourses)
  - Quick review (see announcement at CatCourses):
    - https://classroom.udacity.com/courses/ud219