Fiscal Year 2022

Ver. 2022-12-12a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

2. Instruction Set Architecture and single-cycle processor

www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W831, HyFlex
Mon 13:45-15:25, Thr 13:45-15:25

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

MIPS R3000 32-bit Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

Registers

Registers
R0 - R31
PC
HI
LO

3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	shamt	funct	R format
OP	rs	rt	immediate			I format
OP	jump target (immediate)					J format

MIPS/SPIM Reference Card

CORE INSTRUCTION SET (INCLUDING PSEUDO INSTRUCTIONS)

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd]=R[rs]+R[rt]	(1)	0/20
Add Immediate	addi	I	R[rt]=R[rs]+SignExtImm	(1)(2)	8
Add Imm. Unsigned	addiu	I	R[rt]=R[rs]+SignExtImm	(2)	9
Add Unsigned	addu	R	R[rd]=R[rs]+R[rt]	(2)	0/21
Subtract	sub	R	R[rd]=R[rs]-R[rt]	(1)	0/22
Subtract Unsigned	subu	R	R[rd]=R[rs]-R[rt]		0/23
And	and	R	R[rd]=R[rs]&R[rt]		0/24
And Immediate	andi	I	R[rt]=R[rs]&ZeroExtImm	(3)	c
Nor	nor	R	R[rd]=∼(R[rs]|R[rt])		0/27
Or	or	R	R[rd]=R[rs]|R[rt]		0/25
Or Immediate	ori	I	R[rt]=R[rs]|ZeroExtImm	(3)	d
Xor	xor	R	R[rd]=R[rs]ˆR[rt]		0/26
Xor Immediate	xori	I	R[rt]=R[rs]ˆZeroExtImm		e
Shift Left Logical	sll	R	R[rd]=R[rs]≪shamt		0/00
Shift Right Logical	srl	R	R[rd]=R[rs]≫shamt		0/02
Shift Right Arithmetic	sra	R	R[rd]=R[rs]≫>shamt		0/03
Shift Left Logical Var.	sllv	R	R[rd]=R[rs]≪R[rt]		0/04
Shift Right Logical Var.	srlv	R	R[rd]=R[rs]≫R[rt]		0/06
Shift Right Arithmetic Var.	srav	R	R[rd]=R[rs]≫>R[rt]		0/07
Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09
Move	move	P	R[rd]=R[rs]		
Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
Load Upper Imm.	lui	I	R[rt]={imm,16'b0}		f
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
Load Immediate	li	P	R[rd]=immediate		
Load Address	la	P	R[rd]=immediate		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

REGISTERS

NAME	NMBR	USE	STORE?
$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	Yes
$f0-$f31	0-31	Floating Point Registers	Yes

(1) May cause overflow exception
(2) SignExtImm = {16{immediate[15]},immediate }
(3) ZeroExtImm = {16{1b'0},immediate }
(4) BranchAddr = {14{immediate[15]},immediate,2'b0 }
(4) JumpAddr = {PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

BASIC INSTRUCTION FORMATS, FLOATING POINT 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	immediate				
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	rt	immediate		

MIPS Register Convention and ABI

Name	Register Number	Usage	Preserve on call?
$zero	0	**constant 0 (hardware)**	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	**arguments**	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

ABI (Application Binary Interface)

MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```

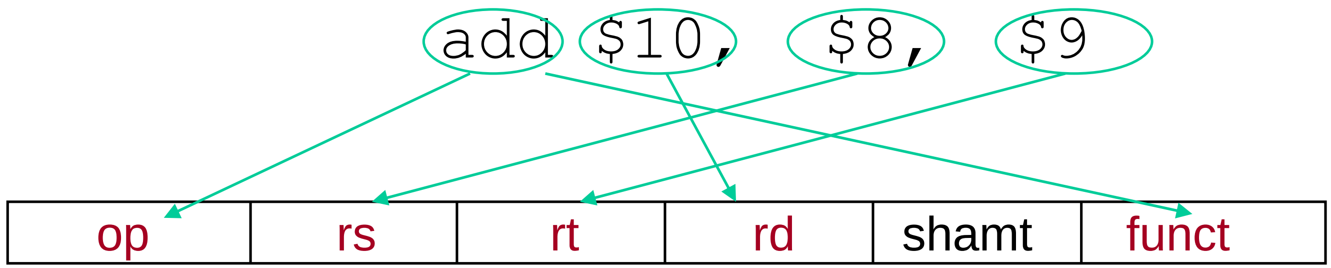
- Each arithmetic instruction performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands

destination ← source1 op source2

- Operand order is fixed (destination first)
- Those operands are all contained in the datapath's register file (`$t0,$s1,$s2`) – indicated by $

Machine Language - Add Instruction

- Instructions, like registers and words, are 32 bits long
- Arithmetic Instruction Format (R format):



op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for shift instructions)
funct	6-bits	function code augmenting the opcode

```
{6'h0, 5'd8, 5'd9, 5'd10, 5'd0, 6'h20} for add $10, $8, $9
```

Exercise

- Compiling a C assignment Using Registers

- `f = ( g + h ) - ( i + j );`
- The variables f, g, h, i, and j are assigned to the registers \$s0, \$s1, \$s2, \$s3, and \$s4, respectively. What is the compiled MIPS code?

MIPS Immediate Instructions

- Small constants are used often in typical code
- Possible approaches?
  - put "typical constants" in memory and load them
  - create hard-wired registers (like $zero) for constants like 1
  - have special instructions that contain constants !

```
addi $sp, $sp, 4   # $sp = $sp + 4
slti $t0, $s2, 15 # $t0 = 1 if $s2<15
```

- Machine format (I format):

op	rs	rt	16 bit immediate

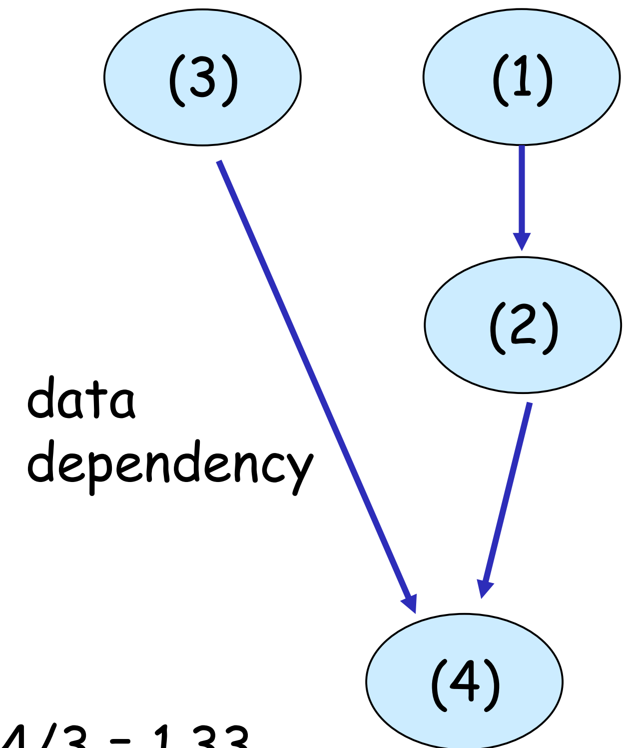
I format

- The constant is kept inside the instruction itself!
  - Immediate format limits values to the range $+2^{15}-1$ to -2^{15}

```
{6'h8, 5'd0, 5'd8, 16'd3} for addi $8, $0, 3
```

Instruction Level Parallelism (ILP)

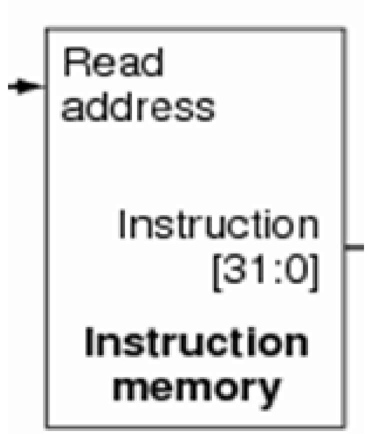
```
add   $8, $3, $5        (1)
addi  $9, $8, 1         (2)
addi  $10, $5, 1        (3)
add   $11, $10, $9      (4)
```



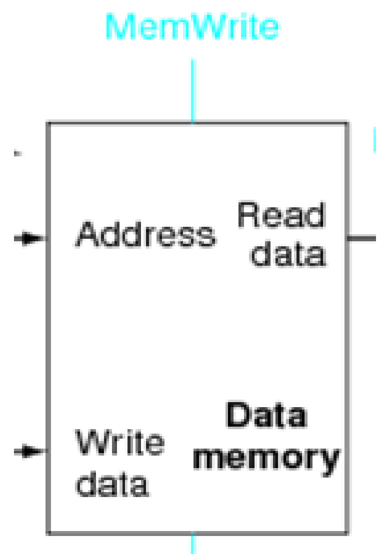
ILP = 4/3 = 1.33

Computer Memory

- Read-only memory (ROM)
- Random-access memory (RAM)

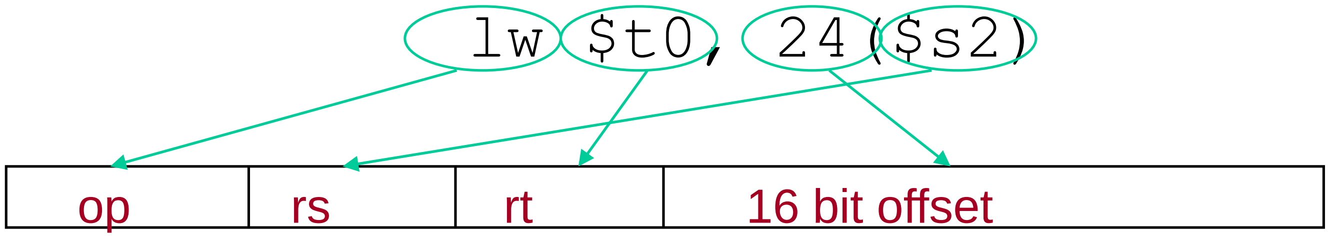


We use 8K word memory.



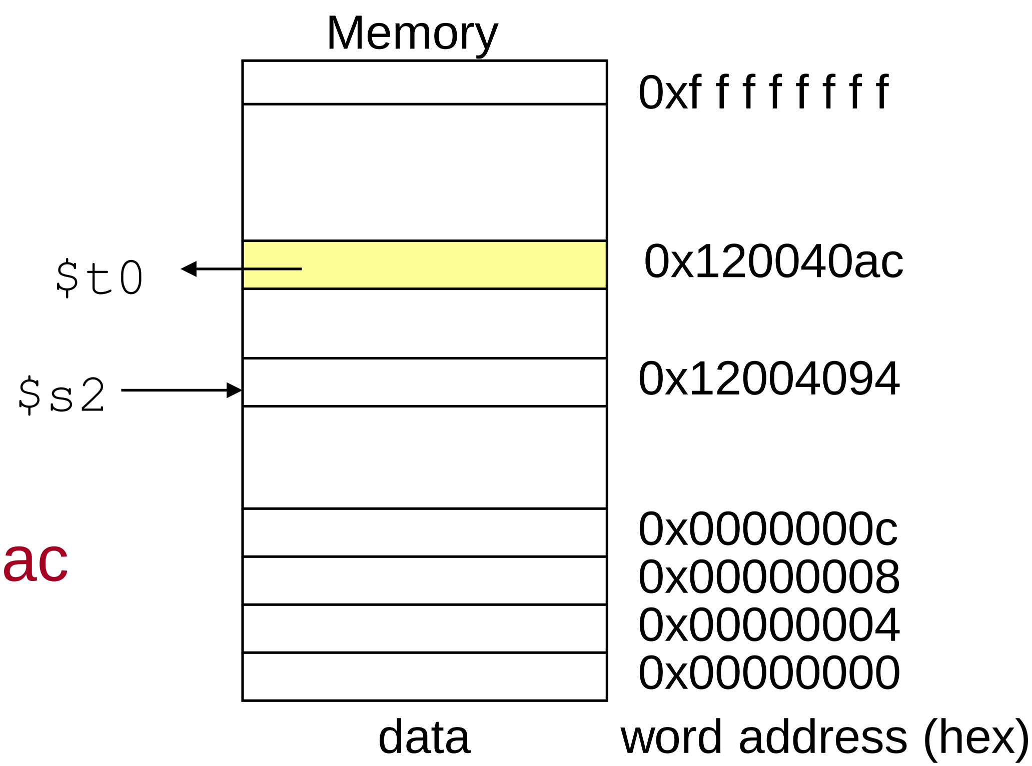
Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



24_{10} + $s2 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 = 0x120040ac



Exercise

- Compiling an Assignment When an Operand Is in Memory

- `g = h + A[8];`
- Let's assume that A is an array of 100 words and the compiler has associated the variable g and h with the registers $s1 and $s2 as before. Let's also assume that the starting address, or base address, of the array is in $s3. Compile this C assignment statement.

this slide is to be used as a whiteboard

MIPS Memory Access Instructions

- MIPS has two basic data transfer instructions for accessing memory

```
lw   $t0, 4($s3)   #load word from memory
sw   $t0, 8($s3)   #store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address
- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value
  - A 16-bit field meaning access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register
  - Note that the offset can be positive or negative

Exercise

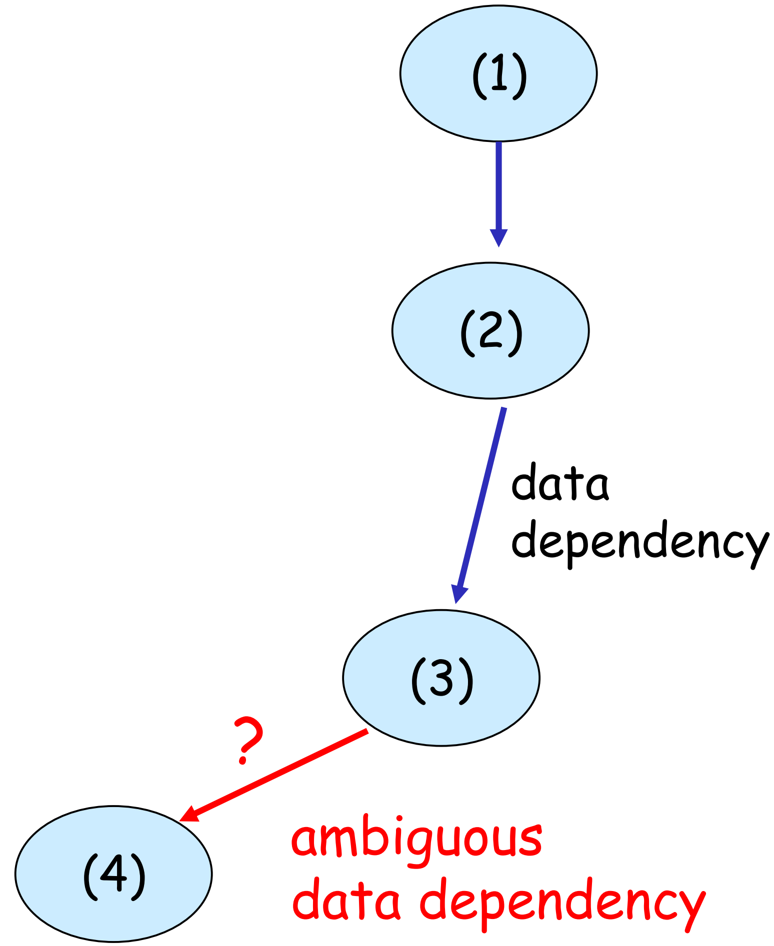
- Compiling Using Load and Store

- `A[12] = h + A[8];`
- Assume variable h is associated with register $s2 and base address of the array *A* is in $s3. What is the MIPS assembly code for the C assignment statement?

this slide is to be used as a whiteboard

Instruction Level Parallelism (ILP)

```
lw    $t0, 32($s3)        (1)
add   $t0, $s2, $t0       (2)
sw    $t0, 48($s3)        (3)
          ?
lw    $t1, 32($s4)        (4)
```



MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl   # go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl   # go to Lbl if $s0=$s1
```

  - Ex: `if (i==j) h = i + j;`

```
    bne $s0, $s1, Lbl1
    add $s3, $s0, $s1
Lbl1:  ...
```

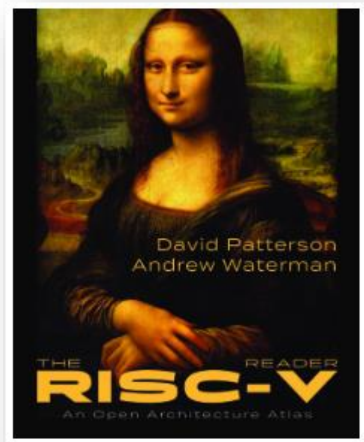
- Instruction Format (I format):

op	rs	rt	16 bit offset

- How is the branch destination address specified?

RISC - Reduced Instruction Set Computer

- RISC philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited addressing modes
  - limited operations
- RISC-I, MIPS, DEC Alpha, ARM, RISC-V, …

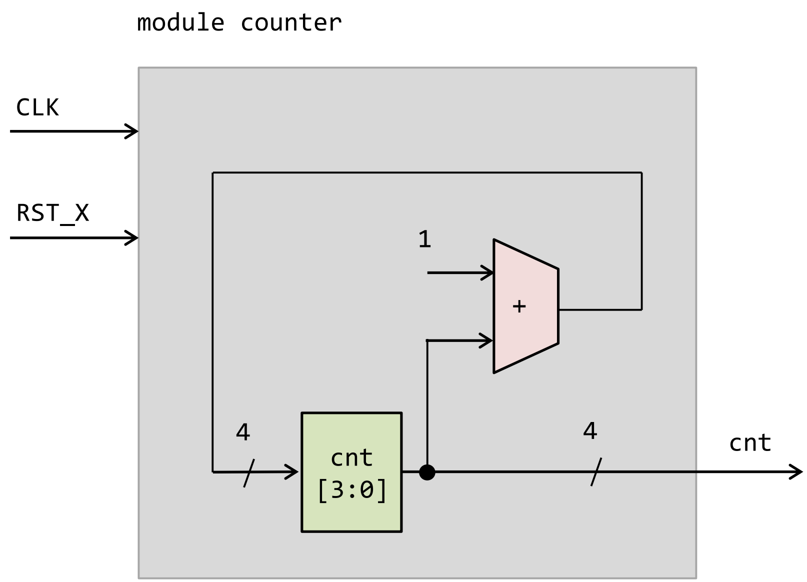


CISC - Complex Instruction Set Computer

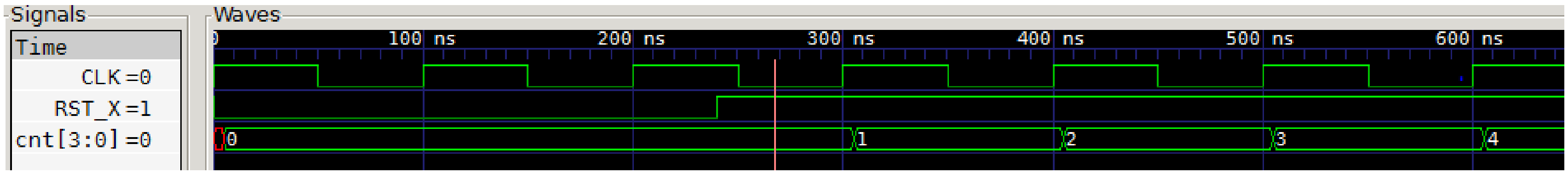
- CISC philosophy
  - ! fixed instruction lengths
  - ! load-store instruction sets
  - ! limited addressing modes
  - ! limited operations
- DEC VAX11, Intel 80x86, …

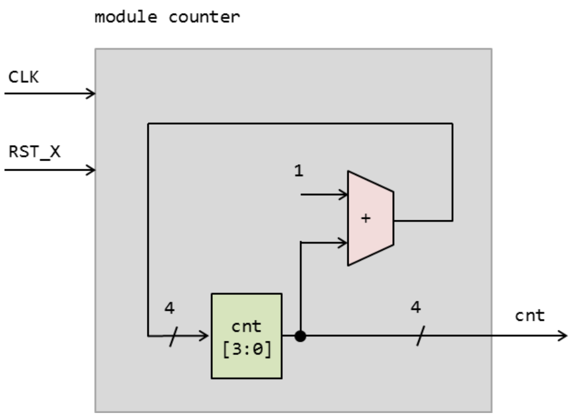
Sample circuit 1

- 4-bit counter
  - synchronous reset
  - negative-logic reset, initialize or reset the value of register cnt to zero if RST_X is low



Sample Verilog HDL Code





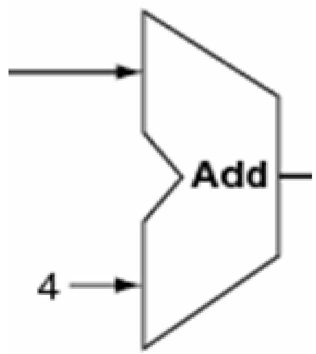
counter.v

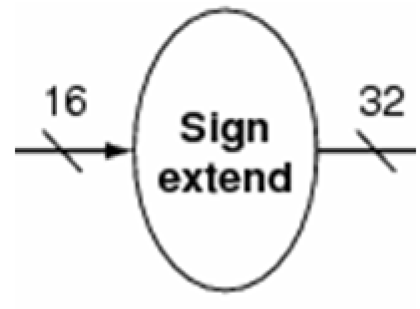
```verilog
module top();
  reg CLK, RST_X;
  wire [3:0] w_cnt;

  initial begin CLK = 1; forever #50 CLK = ~CLK; end
  initial begin RST_X = 0; #240 RST_X = 1; end
  initial #800 $finish();
  initial begin
    $dumpfile("wave.vcd");
    $dumpvars(0, cnt1);
  end
  always @(posedge CLK) $write("cnt1: %d %x¥n", RST_X, w_cnt);

  counter cnt1(CLK, RST_X, w_cnt);
endmodule


module counter(CLK, RST_X, cnt);
  input  wire CLK, RST_X;
  output reg [3:0] cnt;

  always @(posedge CLK) begin
    if(!RST_X) cnt <= #5 0;
    else       cnt <= #5 cnt + 1;
  end
endmodule
```

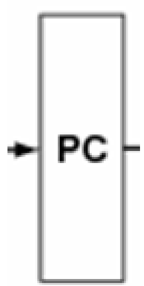
Single-cycle implementation of processors

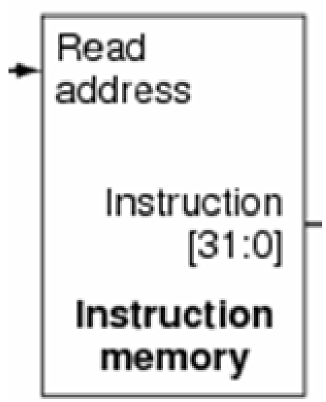
- Single-cycle implementation also called single clock cycle implementation is the implementation in which an instruction is executed in one clock cycle. While easy to understand, it is too slow to be practical.

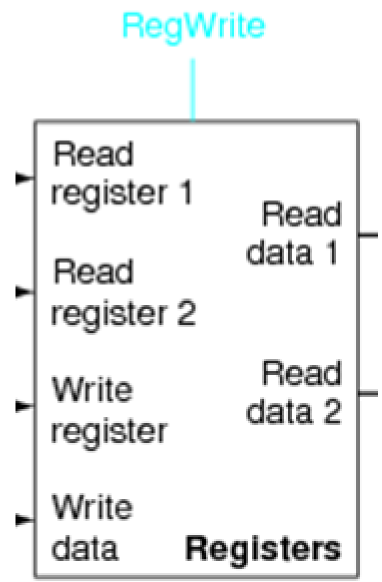
Some building blocks of processor datapath









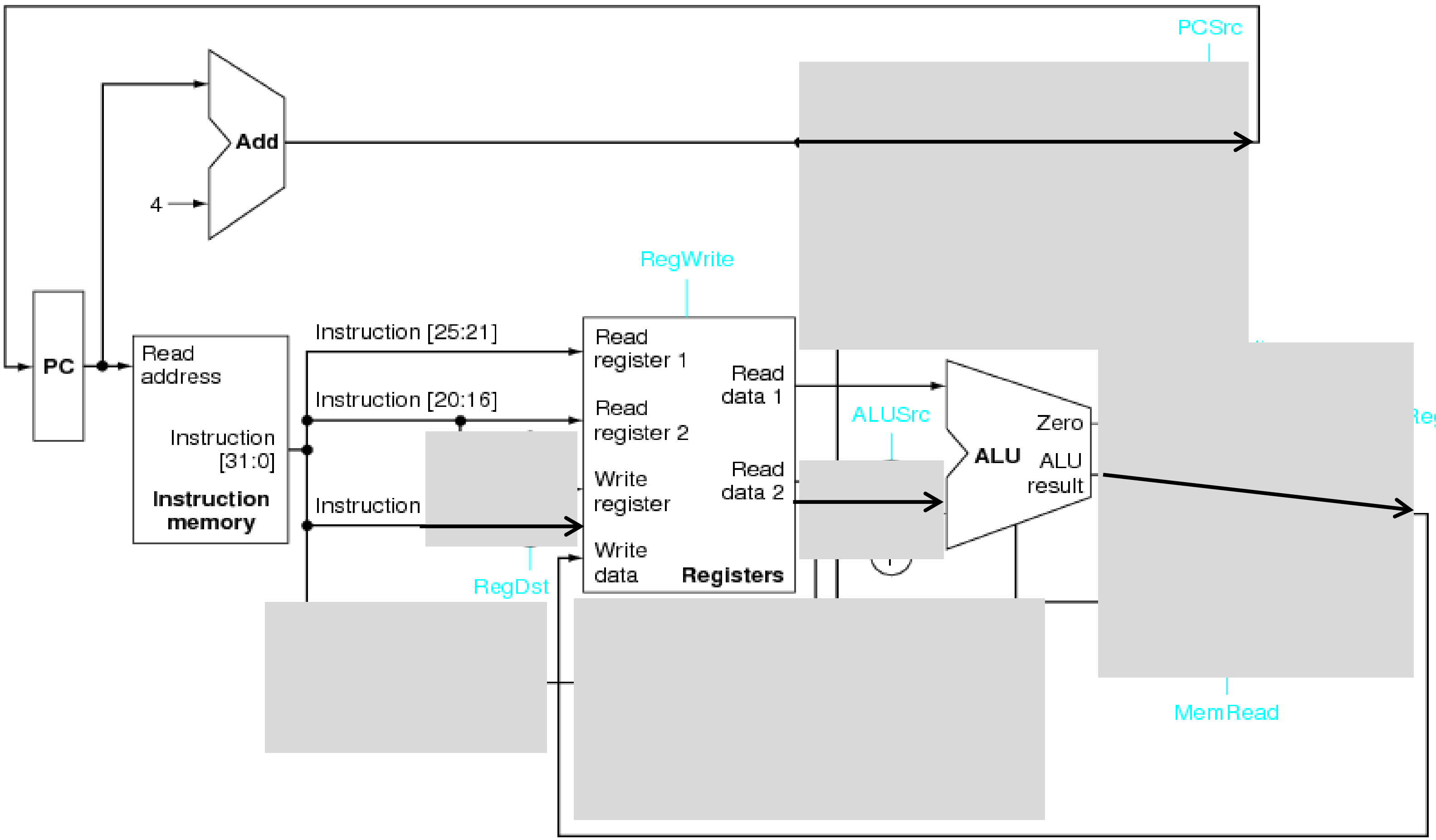


We use 8K word memory.

Datapath of single-cycle processor supporting ADD

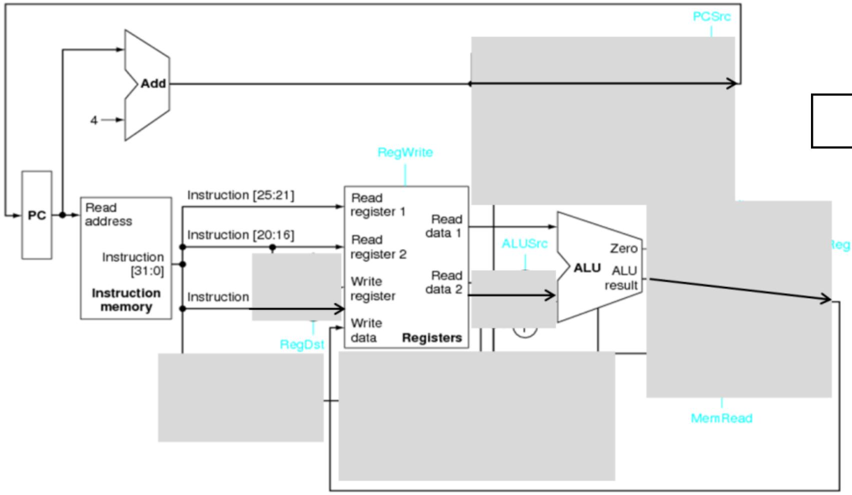
	IR[25:21]	IR[20:16]	IR[15:11]		
op	rs	rt	rd	shamt	funct

0x800 add $t0, $s1, $s2 [add $8, $17, $18]



$17 = 3
$18 = 4

Verilog HDL Code of proc01

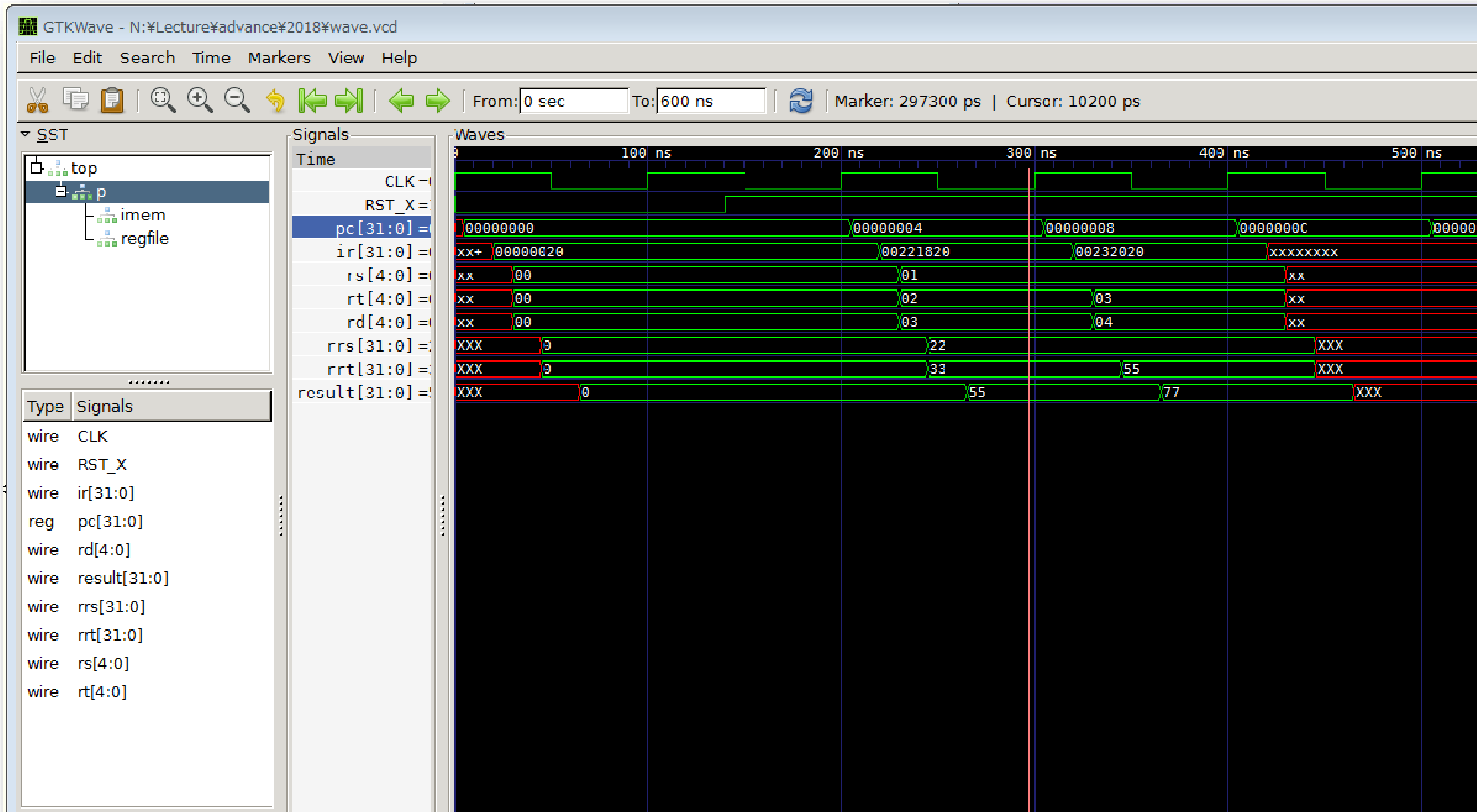


	IR[25:21]	IR[20:16]	IR[15:11]		
op	rs	rt	rd	shamt	funct

```
module PROCESSOR_01(CLK, RST_X);
  input wire CLK, RST_X;

  reg  [31:0] pc;
  wire [31:0] ir;
  wire [31:0] rrs, rrt;

  always @(posedge CLK) pc <= #5 (!RST_X) ? 0 : pc + 4;

  IMEM imem(CLK, pc, ir); /* instruction memory */

  wire [4:0]  #10 rs = ir[25:21];
  wire [4:0]  #10 rt = ir[20:16];
  wire [4:0]  #10 rd = ir[15:11];
  wire [31:0] #20 result = rrs + rrt; /* ALU */

  GPR regfile(CLK, rs, rt, rd, result, 1, rrs, rrt); /* register file */
endmodule
```

proc01.v

Waveform of proc01

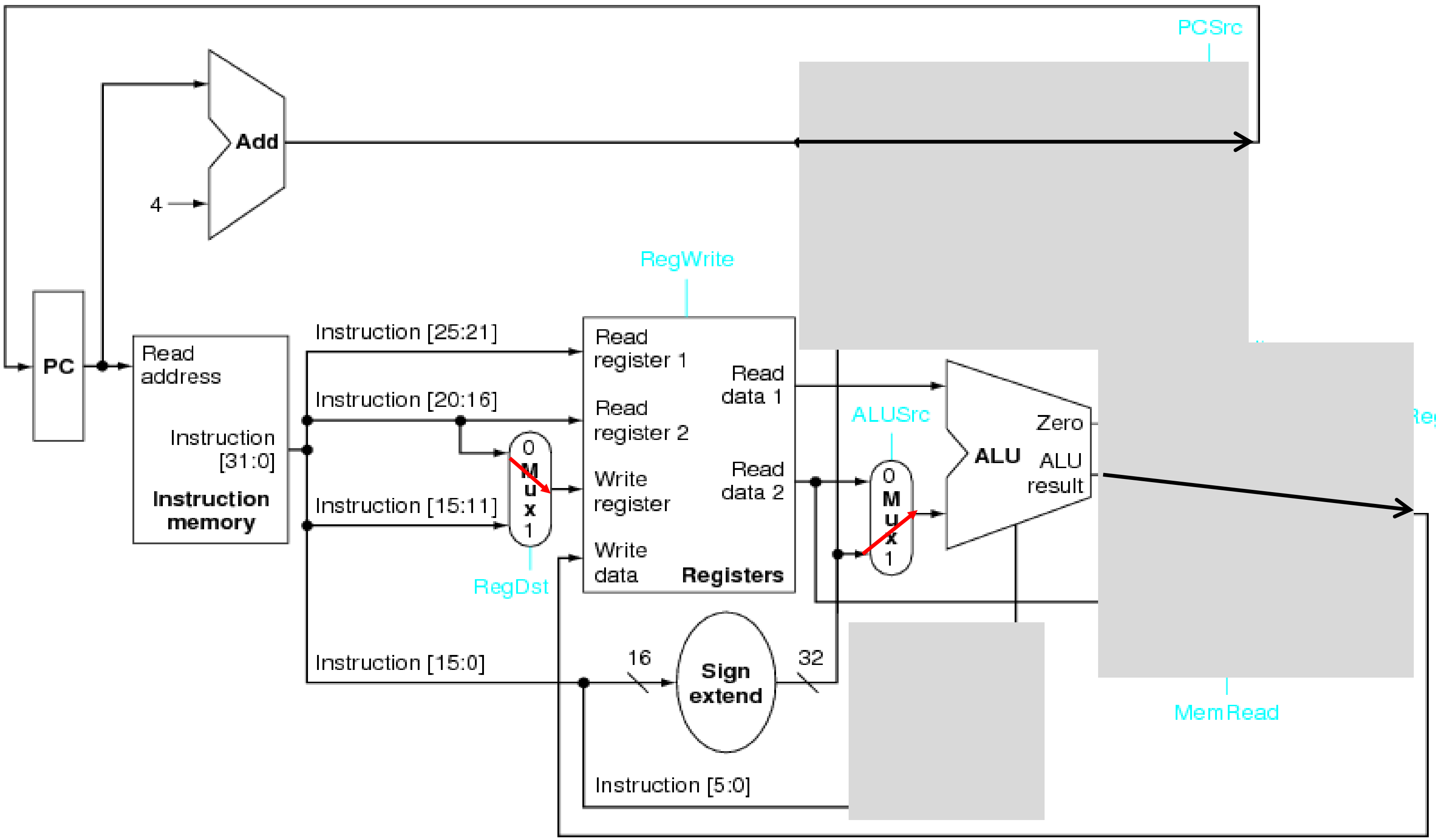


this slide is to be used as a whiteboard

Datapath of processor supporting ADD and ADDI

	IR[25:21]	IR[20:16]		
op	rs	rt	16 bit immediate	I format

0x804 addi $t1, $t0, 3 [addi $9, $8, 3]



$8 = 7

Assignment 1

1. Design a single-cycle processor supporting MIPS add, addi instructions in Verilog HDL. Please download proc01.v from the support page and refer to it.
2. Verify the behavior of designed processor using following assembly code
   - `add  $0, $0, $0   # {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20}`
   - `addi $7, $0, 3    # {6'h8, 5'd0, 5’d7, 16'd3}`
   - `addi $8, $0, 5    # {6'h8, 5'd0, 5’d8, 16'd5}`
   - `add  $9, $7, $8   # {6'h0, 5’d7, 5’d8, 5’d9, 5'd0, 6'h20}`
3. Submit a report printed on A4 paper at the beginning of the next lecture on Monday. Or,
   Submit your report in a PDF file via E-mail (kise [at] c.titech.ac.jp) by the beginning of the next lecture on Monday.
   - The report should include a block diagram, a source code in Verilog HDL, and obtained waveforms of your design.

this slide is to be used as a whiteboard