Alexandria University
Faculty of Engineering
Electrical Engineering Department
Final Exam, June 2016

جامعة الإسكندرية
كلية الهندسة
قسم الهندسة الكهربية
امتحان نهاية الفصل الدراسي الثاني (يونية ٢٠١٦)

Course Title and Code Number: Computer Architectures (CS x35)
Fourth Year (Communications and Electronics)
Time Allowed: 3 hours

اسم المقرر والرقم الكودي له: معماريات الحاسب (CS x35)
السنة الدراسية الرابعة (اتصالات و الكترونيات)
الزمن: ٣ ساعات

Attempt all questions: **(75 marks)**

You can either just draw parts needing modifications in your answer sheet, or you can draw the required modifications on the printed figures in the exam papers and attach them to your answer sheet.

Question 1: **(20 marks)**

a) Modify the single-cycle MIPS processor shown in Figure 1 to implement each of the following instructions. Indicate the changes required to the datapath, control unit, and control signals indicated by Table 1

i. `sll` ii. `sllv` ii. `slti` iv. `jalr`

b) Modify the multi-cycle MIPS processor shown in Figure 2 to implement each of the following instructions. Indicate the changes required to the datapath, control unit, and control FSM indicated by Figure 3. Describe any other changes that are required.

i. `bne` ii. `lb` ii. `j` iv. `jalr`

c) Explain how to extend the pipelined MIPS processor shown in Figure 4 to handle the `addi` and `j` instructions. Give particular attention to how the pipeline is flushed when a jump takes place.

d) Can you modify the Single-cycle processor; Multicycle processor; and Pipelined processors to implement the following instructions. If your answer is yes, indicate the needed modifications to both the datapath and control units. If your answer is no, indicate why and show how these instructions can be implemented using the existing MIPS ISA.

i. `beq rs, rt, rd #if reg(rs)==reg(rt) then PC=reg(rd) else NOP;`

ii.
```
beq rs, imm15:0, Loop
#if reg(rs)==signextend(imm15:0) then PC=BTA(Loop) else NOP;
```

Question 2 **(20 marks)**

a) How many cycles are required to run the following program on the single-cycle and multicycle MIPS processors? What is the CPI of this program on both processors?

```
      add $s0, $0, $0 # i = 0
      add $s1, $0, $0 # sum = 0
      addi $t0, $0, 10 # $t0 = 10
loop:
      slt $t1, $s0, $t0 # if (i < 10), $t1 = 1, else $t1 = 0
      beq $t1, $0, done # if $t1 == 0 (i > = 10), branch to done
      add $s1, $s1, $s0 # sum = sum + i
      addi $s0, $s0, 1 # increment i
      j loop
done:
```

b) How many cycles are required for the pipelined MIPS processor to issue all of the instructions for the program in (a)? What is the CPI of the processor on this program?

c) The pipelined MIPS processor is running the following program. Which registers are being written, and which are being read on the fifth cycle for the processor: (i) without a hazard unit, (ii) having a hazard unit.

```
addi $s1, $s2, 5
sub $t0, $t1, $t2
lw $t3, 15($s1)
sw $t5, 72($t0)
or $t2, $s4, $s5
```

d) A standard benchmark consists of approximately 20% loads, 10% stores, 15% branches, 5% jumps, and 50% R-type instructions. Assume that 30% of the loads are immediately followed by an instruction that uses the result, requiring a stall, and that 25% of the branches are mispredicted, requiring a flush. Assume that jumps always flush the subsequent instruction.

i. Compute the average CPI of the single-cycle, multicycle, and pipelined processors.

ii. Compare the execution time for a program with 10 billion instructions on the three processors. The delay of various circuit elements is shown in Table 2.

iii. Indicate only the most important parameter needing optimization to improve the overall performance of each processor from both the fabrication technology and computer architecture point of views.

Question 3: **(20 marks)**

a) A 16-word cache has the following parameters: b, block size given in numbers of words; S, number of sets; N, number of ways; and A, number of address bits. Consider the following repeating sequence of `lw` addresses (given in hexadecimal)

74 A0 78 38C AC 84 88 8C 7C 34 38 13C 388 18C

Assuming least recently used (LRU) replacement for associative caches, determine the effective miss rate if the sequence is input to the following caches.

i. direct mapped cache, b = 1 word

ii. fully associative cache, b = 2 words

iii. two-way set associative cache, b = 2 words

iv. direct mapped cache, b = 4 words

b) You are building an instruction cache for a MIPS processor. It has a total capacity of $4C = 2^{c+2}$ bytes. It is $N = 2^n$-way set associative ($N \geq 8$), with a block size of $b = 2^{b'}$ bytes ($b \geq 8$). Give your answers to the following questions in terms of these parameters.

i. Which bits of the address are used to select a word within a block?

ii. Which bits of the address are used to select the set within the cache?

iii. How many bits are in each tag?

iv. How many tag bits are in the entire cache?

c) Consider a cache with the following parameters:
N (associativity) = 2, b (block size) = 2 words, W (word size) = 32 bits, C (cache size) = 32 K words, A (address size) = 32 bits. You need to consider only word addresses.

i. Show the tag, set, block offset, and byte offset bits of the address. State how many bits are needed for each field?
ii. What is the size of all the cache tags in bits?
iii. Suppose each cache block also has a valid bit (V) and a dirty bit (D). What is the size of each cache set, including data, tag, and status bits?
iv. Design the cache using the building blocks in Figure 5 and a small number of two-input logic gates. The cache design must include tag storage, data storage, address comparison, data output selection, and any other parts you feel are relevant. Note that the multiplexer and comparator blocks may be any size (n or p bits wide, respectively), but the SRAM blocks must be 16K × 4 bits. Be sure to include a neatly labeled block diagram. You need only design the cache for reads.

d) Consider a virtual memory system that can address a total of 2^{50} bytes. You have unlimited hard drive space, but are limited to 2 GB of semiconductor (physical) memory. Assume that virtual and physical pages are each 4 KB in size.
i. How many bits is the physical and virtual addresses?
ii. What is the number of physical and virtual pages in this system?
iii. How many bits are the virtual and physical page numbers?
iv. How many page table entries will the page table contain?
v. How many bytes long is each page table entry taking the valid bit into consideration?
vi. Sketch the layout of the page table. What is the total size of the page table in bytes?

Question 4: **(15 marks)**

a) Arithmetic mean x and standard deviation σ are defined as

$$\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \qquad \sigma^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \overline{x})^2$$

Assume samples x_n arrive sequentially one at a time. More specifically, each clock cycle sees a new w-bit data item appear. Find a dedicated architecture that computes $\bar{x}$ and σ^2 after N clock cycles and where N is some integer power of two, say 32. Definitions in the above equations suggest one needs to store up to $N-1$ past values of x. Can you make do with less? What mathematical properties do you call on? What is the impact on datapath word width? This is actually an old problem the solution of which has been made popular by early scientific pocket calculators such as the HP-45, for instance. Yet, it nicely shows the difference between a crude and a more elaborate way of organizing a computation.

b) Consider the third-order correlator in Figure 6.
i. To boost performance, try to retime and pipeline the isomorphic architecture without prior reversal of the adder chain. Compare the obtained circuit wit Figure 6(b). Give estimates for datapath resources, cycles per data item, longest path, latency, and control overhead?
ii. Next assume your prime concern is area occupation. What architectures qualify?

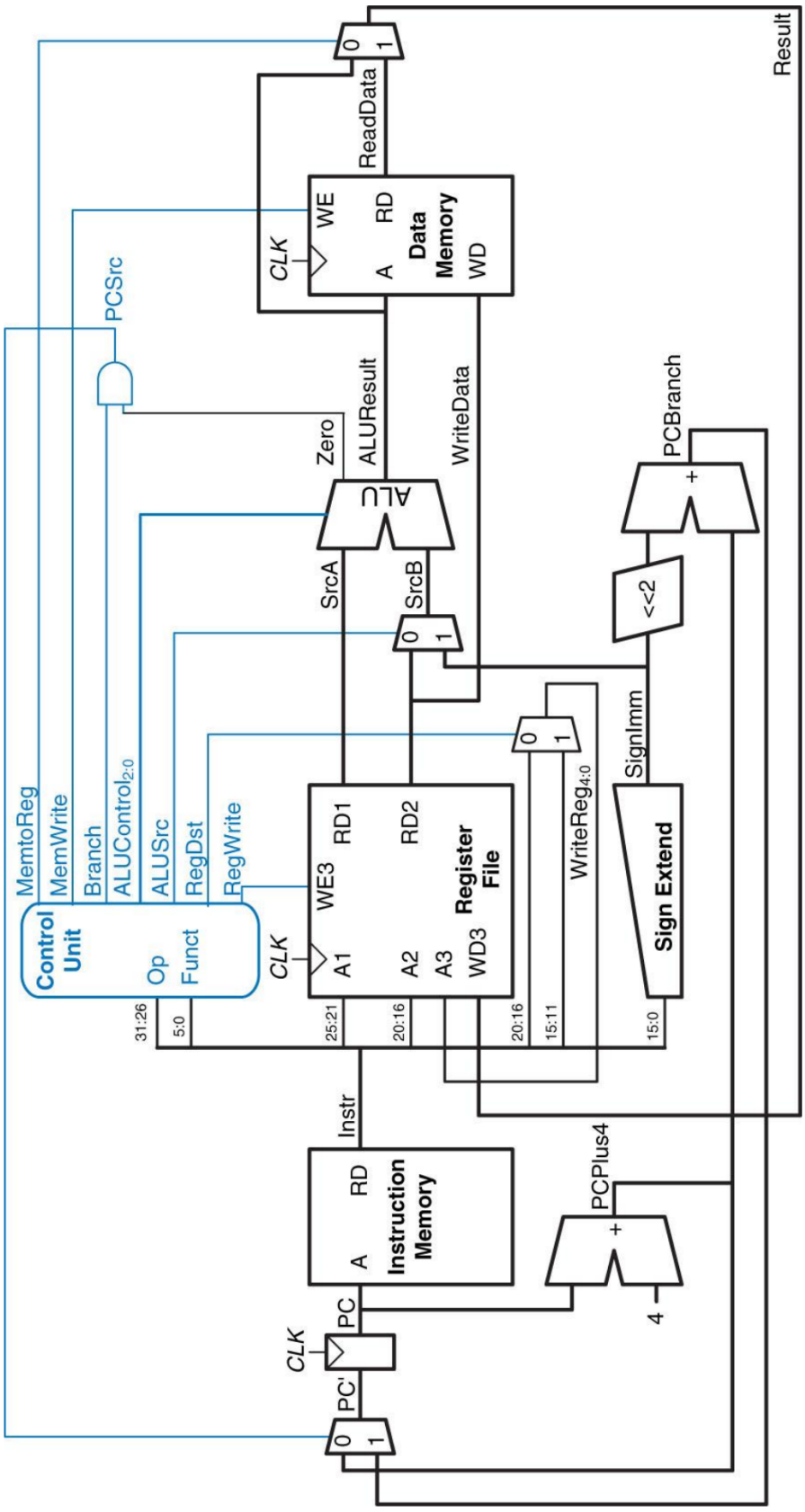


Figure 1: Single-cycle MIPS processor

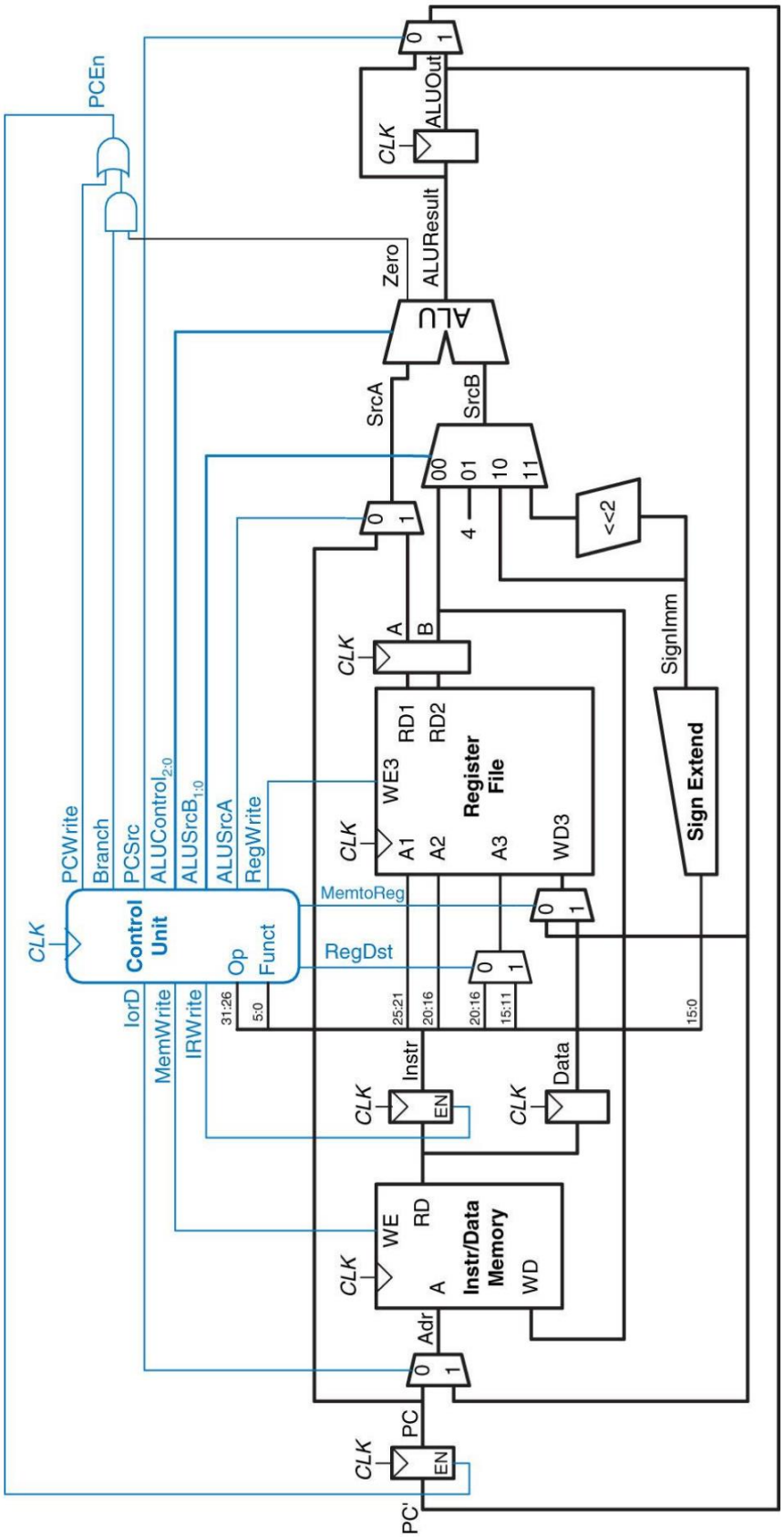


Figure 2: Multicycle MIPS processor

Table 1: Single cycle MIPS processor control signals

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump			
`R-type`	000000	1	1	0	0	0	0	10	0			
`lw`	100011	1	0	1	0	0	1	00	0			
`sw`	101011	0	X	1	0	1	X	00	0			
`beq`	000100	0	X	0	1	0	X	01	0			
`j`	000100	0	X	X	X	0	X	XX	1			

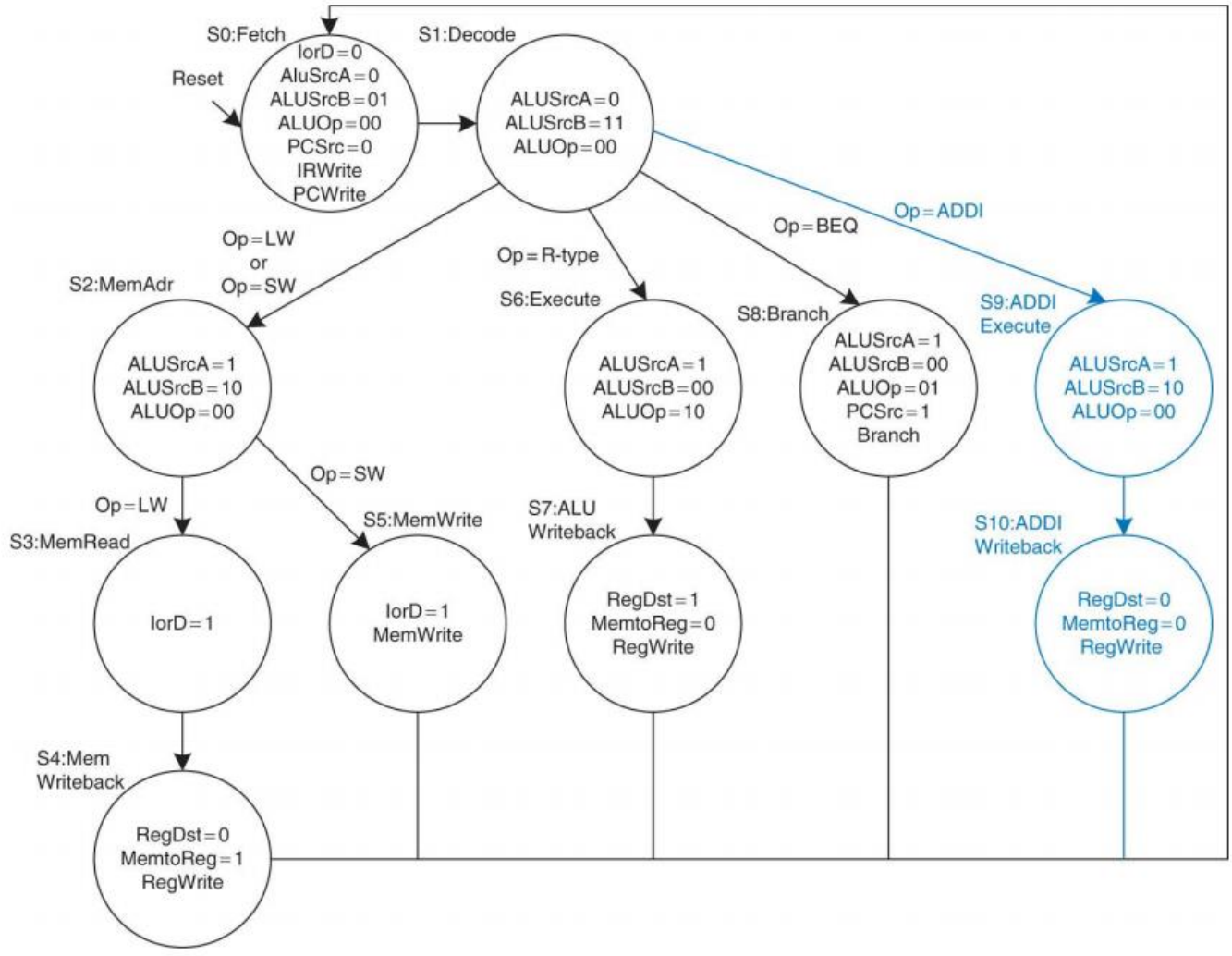


Figure 3: Multicycle MIPS controller FSM

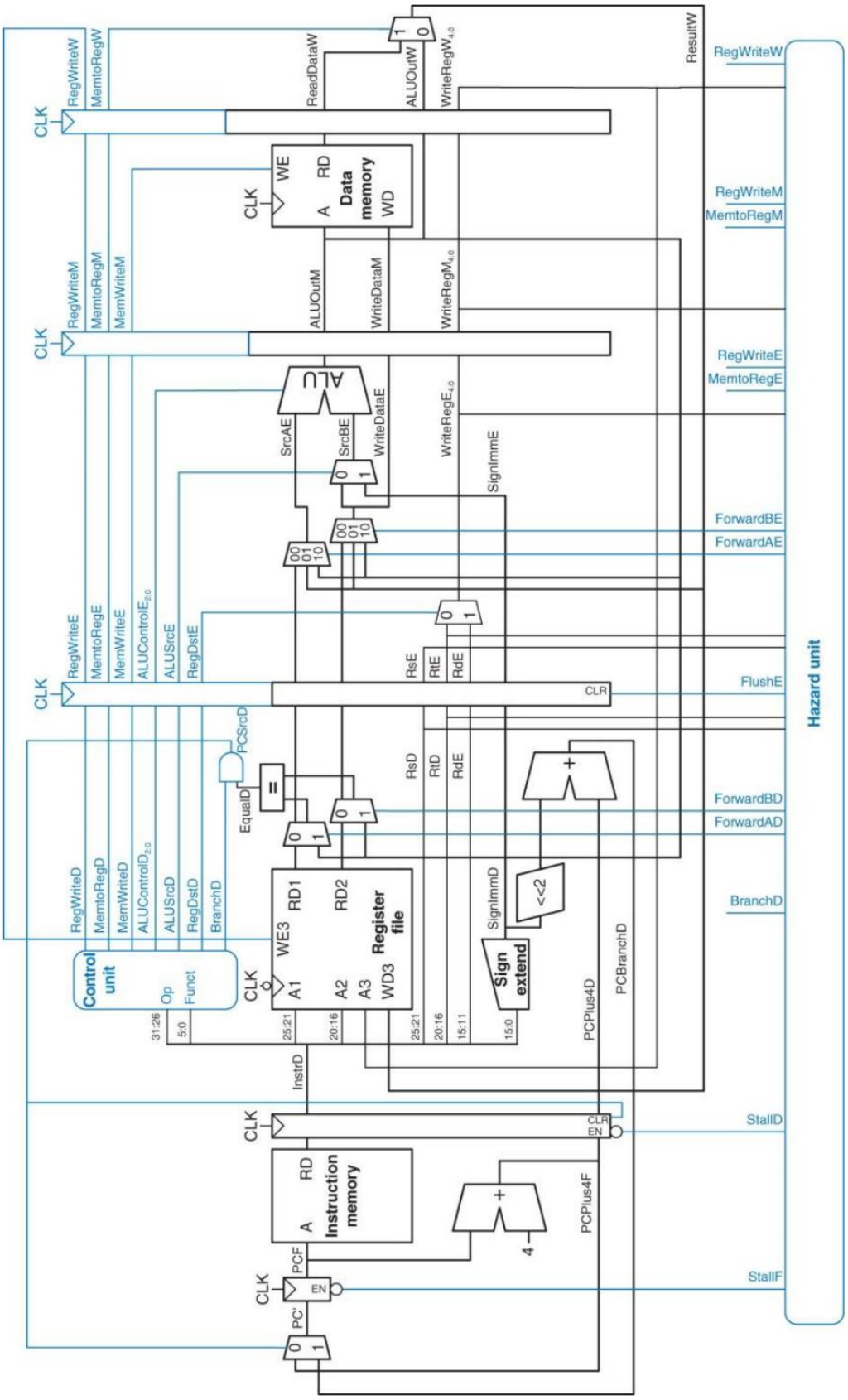


Figure 4: Pipelined MIPS Processor

Table 2: Delays of MIPS circuit elements

Element	Parameter	Delay (ps)
register clk-to-Q	t_{pcq}	20
register setup	t_{setup}	30
multiplexer	t_{mux}	25
ALU	t_{ALU}	350
memory read	t_{mem}	250
register file read	t_{RFread}	150
register file setup	$t_{RFsetup}$	30

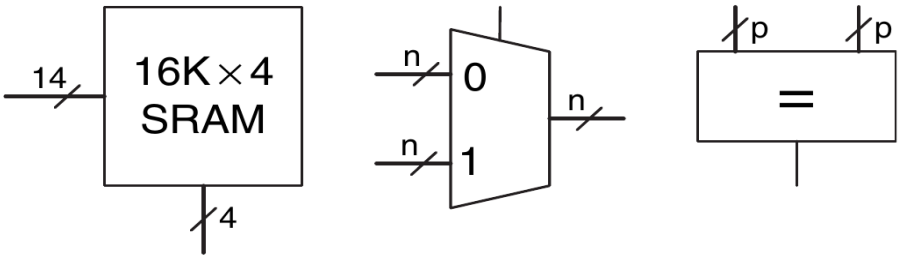
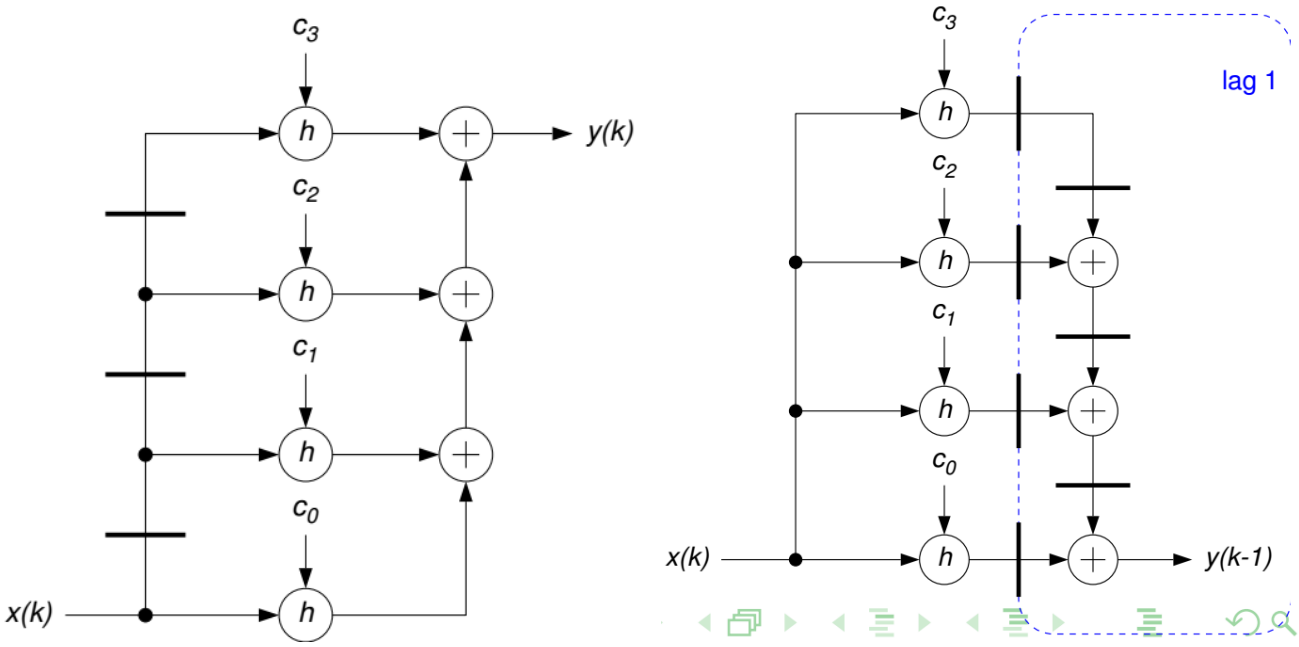


Figure 5: Cache building blocks



(a) isomorphic architecture, (b) retimed, pipelined architecture

Figure 6: Third-order correlator.

MIPS Reference Cheat Sheet

INSTSTRUCTION SET (SUBSET)

Name (format, op, funct)	Syntax	Operation
add (R,0,32)	`add rd,rs,rt`	reg(rd) := reg(rs) + reg(rt);
add immediate (I,8,na)	`addi rt,rs,imm`	reg(rt) := reg(rs) + signext(imm);
add immediate unsigned (I,9,na)	`addiu rt,rs,imm`	reg(rt) := reg(rs) + signext(imm);
add unsigned (R,0,33)	`addu rd,rs,rt`	reg(rd) := reg(rs) + reg(rt);
and (R,0,36)	`and rd,rs,rt`	reg(rd) := reg(rs) & reg(rt);
and immediate (I,12,na)	`andi rt,rs,imm`	reg(rt) := reg(rs) & zeroext(imm);
branch on equal (I,4,na)	`beq rs,rt,label`	if reg(rs) == reg(rt) then PC = BTA else NOP;
branch on not equal (I,5,na)	`bne rs,rt,label`	if reg(rs) != reg(rt) then PC = BTA else NOP;
jump and link register (R,0,9)	`jalr rs`	$ra := PC + 4; PC := reg(rs);
jump register (R,0,8)	`jr rs`	PC := reg(rs);
jump (J,2,na)	`j label`	PC := JTA;
jump and link (J,3,na)	`jal label`	$ra := PC + 4; PC := JTA;
load byte (I,32,na)	`lb rt,imm(rs)`	reg(rt) := signext(mem[reg(rs) + signext(imm)]$_{7:0}$);
load byte unsigned (I,36,na)	`lbu rt,imm(rs)`	reg(rt) := zeroext(mem[reg(rs) + signext(imm)]$_{7:0}$);
load upper immediate (I,14,na)	`lui rt,imm`	reg(rt) := concat(imm, 16 bits of 0);
load word (I,35,na)	`lw rt,imm(rs)`	reg(rt) := mem[reg(rs) + signext(imm)];
multiply, 32-bit result (R,28,2)	`mul rd,rs,rt`	reg(rd) := reg(rs) * reg(rt);
nor (R,0,39)	`nor rd,rs,rt`	reg(rd) := not(reg(rs) | reg(rt));
or (R,0,37)	`or rd,rs,rt`	reg(rd) := reg(rs) | reg(rt);
or immediate (I,13,na)	`ori rt,rs,imm`	reg(rt) := reg(rs) | zeroext(imm);
set less than (R,0,42)	`slt rd,rs,rt`	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than unsigned (R,0,43)	`sltu rd,rs,rt`	reg(rd) := if reg(rs) < reg(rt) then 1 else 0;
set less than immediate (I,10,na)	`slti rt,rs,imm`	reg(rt) := if reg(rs) < signext(imm) then 1 else 0;
set less than immediate unsigned (I,11,na)	`sltiu rt,rs,imm`	reg(rt) := if reg(rs) < signext(imm) then 1 else 0;
shift left logical (R,0,0)	`sll rd,rt,shamt`	reg(rd) := reg(rt) << shamt;
shift left logical variable (R,0,4)	`sllv rd,rt,rs`	reg(rd) := reg(rt) << reg(rs$_{4:0}$);
shift right arithmetic (R,0,3)	`sra rd,rt,shamt`	reg(rd) := reg(rt) >>> shamt;
shift right logical (R,0,2)	`srl rd,rt,shamt`	reg(rd) := reg(rt) >> shamt;
shift right logical variable (R,0,6)	`srlv rd,rt,rs`	reg(rd) := reg(rt) >> reg(rs$_{4:0}$);
store byte (I,40,na)	`sb rt,imm(rs)`	mem[reg(rs) + signext(imm)]$_{7:0}$:= reg(rt)$_{7:0}$;
store word (I,43,na)	`sw rt,imm(rs)`	mem[reg(rs) + signext(imm)] := reg(rt);
subtract (R,0,34)	`sub rd,rs,rt`	reg(rd) := reg(rs) - reg(rt);
subtract unsigned (R,0,35)	`subu rd,rs,rt`	reg(rd) := reg(rs) - reg(rt);
xor (R,0,38)	`xor rd,rs,rt`	reg(rd) := reg(rs) ^ reg(rt);
xor immediate (I,14,na)	`xori rt,rs,imm`	reg(rt) := rerg(rs) ^ zeroext(imm);

Definitions

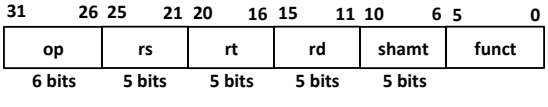
- Jump to target address: JTA = concat((PC + 4)$_{31:28}$, address(label), 00_2)
- Branch target address: BTA = PC + 4 + imm * 4

Clarifications

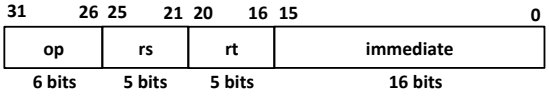
- All numbers are given in decimal form (base 10).
- Function signext(x) returns a 32-bit sign extended value of x in two's complement form.
- Function zeroext(x) returns a 32-bit value, where zero are added to the most significant side of x.
- Function concat(x, y, …, z) concatenates the bits of expressions x, y, …, z.
- Subscripts, for instance $X_{8:2}$, means that bits with index 8 to 2 are spliced out of the integer X.
- Function *address*(x) means the address of label x.
- NOP and na means "no operation" and "not applicable", respectively.
- shamt is an abbreviation for "shift amount", i.e. how much bit shifting that should be done.

INSTRUCTION FORMAT

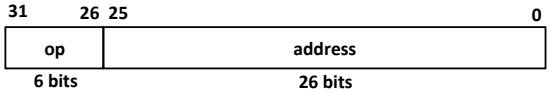
R-Type



I-Type



J-Type



REGISTERS

Name	Number	Description
$zero	0	constant value 0
$at	1	assembler temp
$v0	2	function return
$v1	3	function return
$a0	4	argument
$a1	5	argument
$a2	6	argument
$a3	7	argument
$t0	8	temporary value
$t1	9	temporary value
$t2	10	temporary value
$t3	11	temporary value
$t4	12	temporary value
$t5	13	temporary value
$t6	14	temporary value
$t7	15	temporary value
$s0	16	saved temporary
$s1	17	saved temporary
$s2	18	saved temporary
$s3	19	saved temporary
$s4	20	saved temporary
$s5	21	saved temporary
$s6	22	saved temporary
$s7	23	saved temporary
$t8	24	temporary value
$t9	25	temporary value
$k0	26	reserved for OS
$k1	27	reserved for OS
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

MIPS Reference Cheat Sheet

By David Broman
KTH Royal Institute of Technology

If you find any errors or have any feedback on this document, please send me an email: dbro@kth.se

Version 1.0, December 19, 2014