Alexandria University	جامعة الإسكندرية
Faculty of Engineering	كلية الهندسة
Electrical Engineering Department	قسم الهندسة الكهربية
Final Exam, June 2015	امتحان نهاية الفصل الدراسي الثاني (يونية ٢٠١٥)
Course Title and Code Number:	اسم المقرر والرقم الكودي له:
Computer Architectures (CS x35)	معماريات الحاسب (CS x35)
Fourth Year (Communications and Electronics)	السنة الدراسية الرابعة (اتصالات و الكترونيات)
Time Allowed: 3 hours	الزمن: ٣ ساعات

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

Question 1: **(19 marks)**

a) Implement the following high-level code segments using the `slt` instruction. Assume the integer variables `g` and `h` are in registers `$s0` and `$s1`, respectively.
(**MIPS Instruction Set Summary is given in page 8**)

```
 i.   if (g > h)          ii.   if (g >= h)       iii.   if (g <= h)
         g = g + h;                g = g + 1;               g = 0;
      else                      else                     else
         g = g - h;                h = h - 1;               h = 0;
```

b) Write a MIPS assembly program equivalent to the following pseudo-instructions. If necessary, you can use register `$t0` to memorize intermediary values. No other register can be used.

i. `add ($s0),$s1,($s2) #mem[$s0]=$s1+mem[$s2]`
This MIPS instruction does not exist, because it uses an addressing mode not supported by RISC processors.

ii. `SWAP $s0  # bits 31-16 <-> bits 15-0`
This instruction allows us to swap the 16 most significant bits with the 16 least significant ones of a 32-bit word.

iii. `PUSH $s0`
This instruction is not a MIPS instruction either. It decrements the stack pointer (SP), then saves `$s0` at this address.

c) Consider the following high-level function.

```
// C code
int f(int n, int k) {
  int b;

  b = k + 2;
  if (n == 0) b = 10;
  else b = b + (n * n) + f(n − 1, k + 1);
  return b * k;
}
```

i. Translate the high-level function `f` into MIPS assembly language. Pay particular attention to properly saving and restoring registers across function calls and using the MIPS preserved register conventions. Clearly comment your code. You can use the MIPS `mul` instruction. The function starts at instruction address 0x00400100. Keep local variable `b` in `$s0`.

ii. Step through your function from part (a) by hand for the case of `f(2,4)`. Draw a picture showing stack memory contents. Write the register name and data value stored at each location in the stack and keep track of the stack pointer value (`$sp`). Clearly mark each stack frame. You might also find it useful to keep track of the values in `$a0`, `$a1`, `$v0`, and `$s0` throughout execution. Assume that when f is called, `$s0` = 0xABCD and `$ra = 0x400004`. What is the final value of `$v0`?

Question 2 **(24 marks)**

a) Modify the multicycle MIPS processor to implement one of the following instructions. Indicate the changes to the datapath and name any new control signals. Show the changes to the controller FSM and describe any other changes that are required. (**Add your modifications to Figure 1 and 2 and attach the paper containing pages 5 and 6 to your answer paper**).

   i. `xori`   ii. `jr`   iii. `bne`   iv. `lb`

b) What is the CPI of the redesigned multicycle MIPS? Use delay Table 1 and the following and instruction mix: The **SPECINT2000 benchmark** consists of approximately 25% loads, 10% stores, 11% branches, 2% jumps, and 52% R-type instructions.

c) Explain with examples two data and control hazards in a five-stage MIPS pipelined processor. Show how to handle these hazards at both compile- and run-time.

   (**The Pipelined MIPS processor architecture is shown in Figure 3**)

Table 1: Delay of processor elements

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20
Equality comparator	t_{eq}	40
AND gate	t_{AND}	15
Memory write	$T_{memwrite}$	220
Register file write	$t_{RFwrite}$	100 ps

d) Suppose the MIPS pipelined processor is divided into 10 stages of 400 ps each, including sequencing overhead. Assume the instruction mix of the SPECINT2000 benchmark. Also assume that 50% of the loads are immediately followed by an instruction that uses the result, requiring six stalls, and that 30% of the branches are mispredicted. The target address of a branch or jump instruction is not computed until the end of the second stage. Calculate the average CPI and execution time of computing 100 billion instructions from the SPECINT2000 benchmark for this 10-stage pipelined processor.

Question 3: **(18 marks)**

a) 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?

b) 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 4 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.

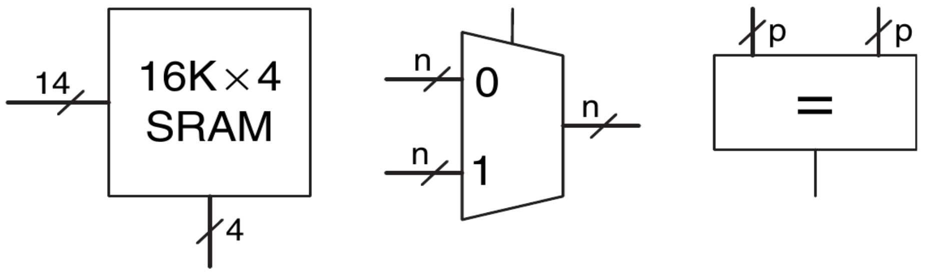

Figure 4: Building blocks

c) Consider a virtual memory system that can address a total of 2^{32} bytes. You have unlimited hard drive space, but are limited to only 8 MB of semiconductor (physical) memory. Assume that virtual and physical pages are each 4 KB in size.
   i. How many bits is the physical address, virtual and physical page numbers?
   ii. What is the maximum number of virtual pages in the system?
   iii. How many physical pages are in the system?
   iv. Suppose that you come up with a direct mapped scheme that maps virtual pages to physical pages. The mapping uses the least significant bits of the virtual page number to determine the physical page number. How many virtual pages are mapped to each physical page? Why is this "direct mapping" a bad plan?
   v. Clearly, a more flexible and dynamic scheme for translating virtual addresses into physical addresses is required than the one described in part (iv). Suppose you use a page table to store mappings (translations from virtual page number to physical page number). How many page table entries will the page table contain?
   vi. Sketch the layout of the page table. What is the total size of the page table in bytes?

Question 4: **(14 marks)**

a) Computationally efficient approximations for the magnitude function are presented in Table 3. Give three alternative architectures that implement the algorithm and compare them in terms of datapath resources, cycles per data item, longest path, and control overhead. Assume input data remain valid as long as you need them, but plan for a registered output. Begin by drawing the DDG.

Table 3: Approximations for computing magnitudes

Name	aka	Formula
lesser	$\ell^{-\infty}$-norm	$l = \min(\|a\|, \|b\|)$
sum	ℓ^{1}-norm	$s = \|a\| + \|b\|$
magnitude (reference)	ℓ^{2}-norm	$m = \sqrt{a^2 + b^2}$
greater	ℓ^{∞}-norm	$g = \max(\|a\|, \|b\|)$
approximation 1		$m \approx m_1 = \frac{3}{8}s + \frac{5}{8}g$
approximation 2 [35]		$m \approx m_2 = \max(g, \frac{7}{8}g + \frac{1}{2}l)$

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?

Good Luck

Dr. Mohammed Morsy

Answer Question 2(a) in this page and attach (staple) it to the answer paper
Do not write your personal data (name or seat number) on this paper

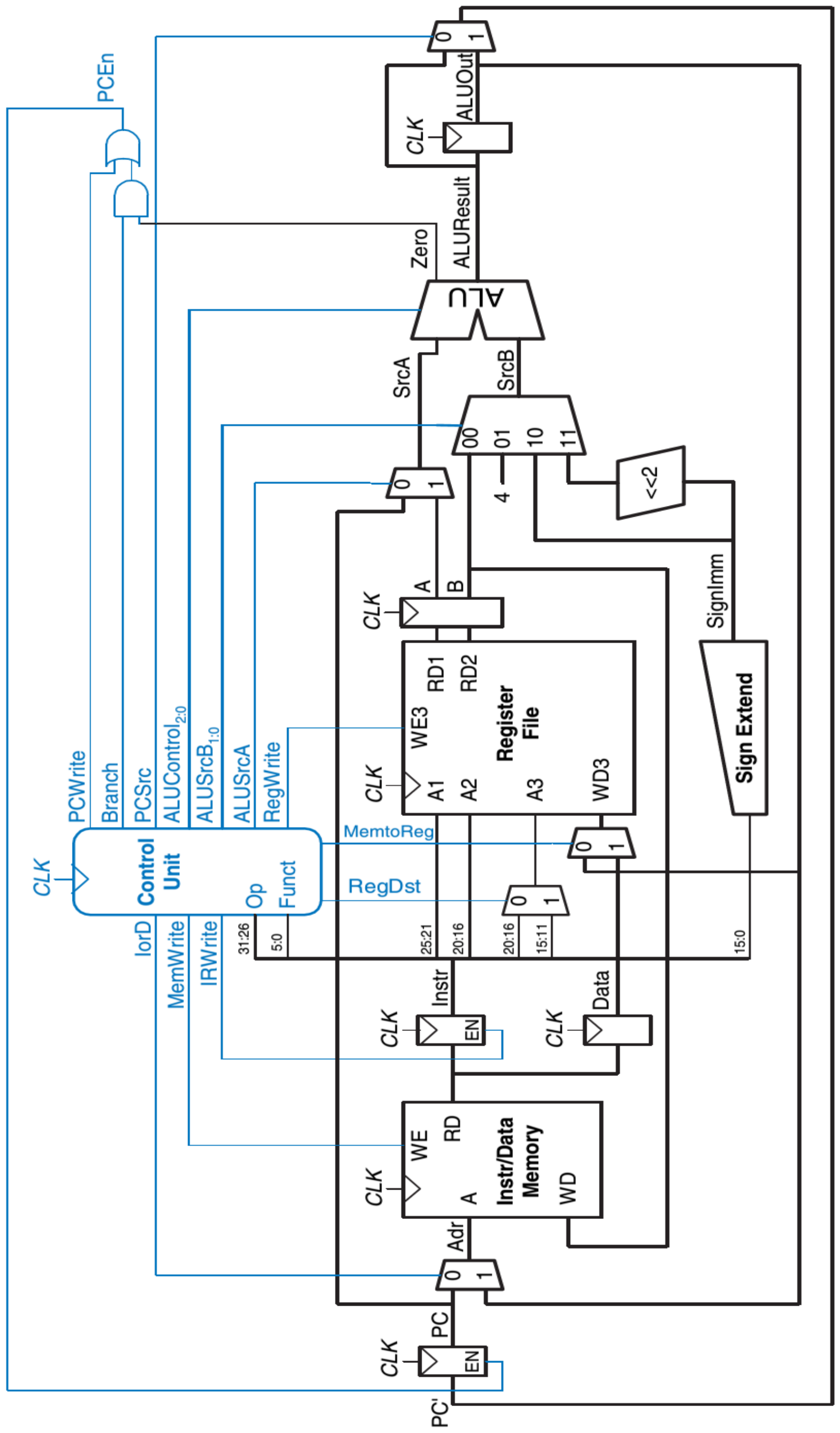

Figure 1: MIPS multicycle processor

Answer Question 2(a) in this page and attach (staple) it to the answer paper
Do not write your personal data (name or seat number) on this paper

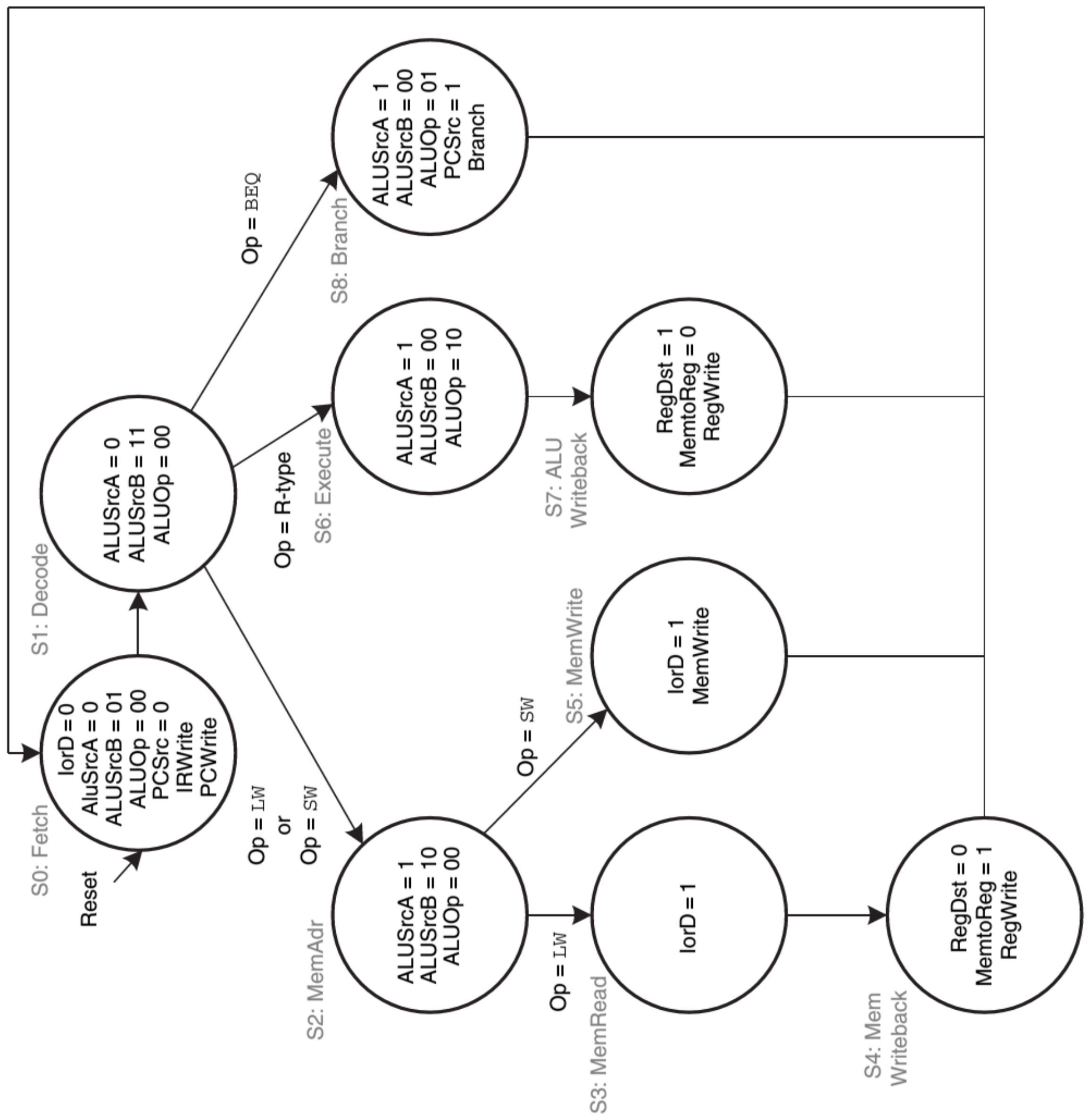

Figure 2: MIPS multicycle processor FSM

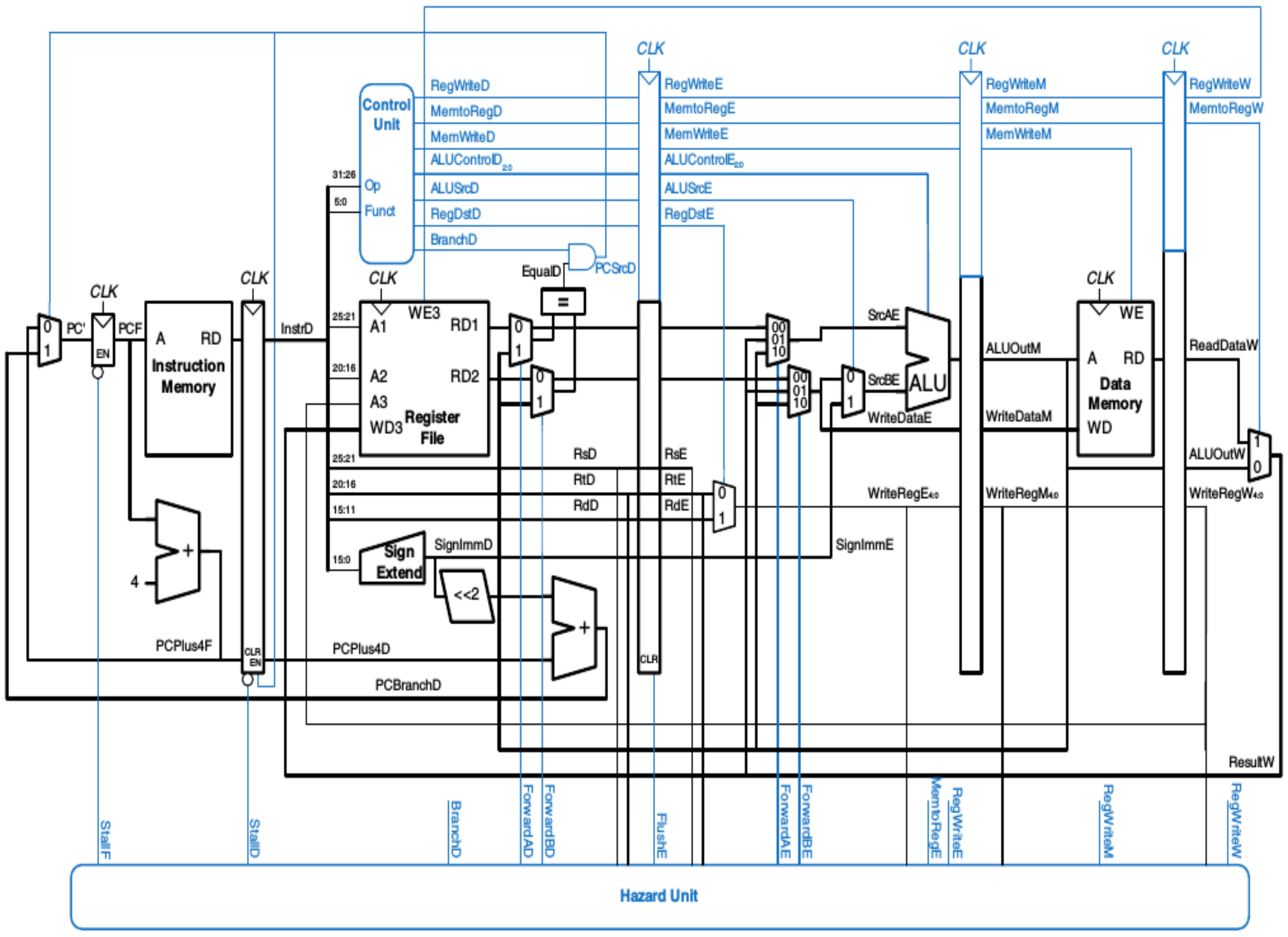

Figure 3: Pipelined MIPS

The MIPS Processor Instruction Format

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

MIPS Instruction Set Summary (Subset)

Opcodes	Example Assembly	Semantics
add	`add $1, $2, $3`	`$1 = $2 + $3`
sub	`sub $1, $2, $3`	`$1 = $2 - $3`
add immediate	`addi $1, $2, 100`	`$1 = $2 + 100`
add unsigned	`addu $1, $2, $3`	`$1 = $2 + $3`
subtract unsigned	`subu $1, $2, $3`	`$1 = $2 - $3`
add imm. Unsigned	`addiu $1, $2, 100`	`$1 = $2 + 100`
multiply	`mult $2, $3`	`hi, lo = $2 * $3`
multiply unsigned	`multu $2, $3`	`hi, lo = $2 * $3`
divide	`div $2, $3`	`lo = $2/$3, hi = $2 mod $3`
divide unsigned	`divu $2, $3`	`lo = $2/$3, hi = $2 mod $3`
move from hi	`mfhi $1`	`$1 = hi`
move from low	`mflo $1`	`$1 = lo`
and	`and $1, $2, $3`	`$1 = $2 & $3`
or	`or $1, $2, $3`	`$1 = $2 | $3`
and immediate	`andi $1, $2, 100`	`$1 = $2 & 100`
or immediate	`ori $1, $2, 100`	`$1 = $2 | 100`
shift left logical	`sll $1, $2, 10`	`$1 = $2 << 10`
shift right logical	`srl $1, $2, 10`	`$1 = $2 >> 10`
load word	`lw $1, $2(100)`	`$1 = ReadMem32($2 + 100)`
store word	`sw $1, $2(100)`	`WriteMem32($2 + 100, $1)`
load halfword	`lh $1, $2(100)`	`$1 = SignExt(ReadMem16($2 + 100))`
store halfword	`sh $1, $2(100)`	`WriteMem16($2 + 100, $1)`
load byte	`lb $1, $2(100)`	`$1 = SignExt(ReadMem8($2 + 100))`
store byte	`sb $1, $2(100)`	`WriteMem8($2 + 100, $1)`
load upper immediate	`lui $1, 100`	`$1 = 100 << 16`
branch on equal	`beq $1, $2, Label`	`if ($1 == $2) goto Label`
branch on not equal	`bne $1, $2, Label`	`if ($1 != $2) goto Label`
set on less than	`slt $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`slti $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
set on less than unsigned	`sltu $1, $2, $3`	`if ($2 < $3) $1 = 1 else $1 = 0`
set on less than immediate	`sltui $1, $2, 100`	`if ($2 < 100) $1 = 1 else $1 = 0`
jump	`j Label`	`goto Label`
jump register	`jr $31`	`goto $31`
jump and link	`jal Label`	`$31 = PC + 4; goto Label`