Digital Logic Systems

Recitation 13: The ISA of a Simplified DLX

Guy Even   Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

January 26, 2012

Instruction formats

Every instruction in the instruction set of the simplified DLX is represented by a single word. There are two instruction formats: I-type and R-type.

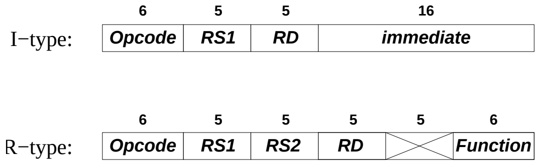

Figure: Instruction formats of the simplified DLX. (Bits are ordered in descending order; namely, the leftmost bit is in position [31] and the rightmost bit is in position [0].)

I-type Instructions

Load/Store	Semantics
lw RD RS1 imm	RD := M[sext(imm)+RS1]
sw RD RS1 imm	M[sext(imm)+RS1] := RD

Example

The assembly instruction lw 7 0 15 means copy the word stored in the memory in address $\langle \mathrm{R0} \rangle + 15$ to $\mathrm{R7}$. Note that ‘0’ stands for $\mathrm{R0}$, which, as we shall see later, always stores the value ‘0’. The assembly instruction sw 7 4 0 means copy the word stored in register $\mathrm{R7}$ to the memory in address $\langle \mathrm{R4} \rangle + 0$.

I-type Instructions (cont.)

Instruction	Semantics
addi RD RS1 imm	RD := RS1 + sext(imm)

Example

The assembly instruction addi 1 1 1 means increment the contents of register $\mathrm{R}1$ by 1.

R-type Instructions

Instruction	Semantics
sll RD RS1	RD := RS1 $<<$ 1
srl RD RS1	RD := RS1 $>>$ 1

Example

The assembly instruction sll 4 8 means: logically shift the contents of register $\mathrm{R}8$ by one position to the left, and store the shifted word in register $\mathrm{R}4$.
The assembly instruction srl 4 8 means: logically shift the contents of register $\mathrm{R}8$ by one position to the right, and store the shifted word in register $\mathrm{R}4$.

R-type Instructions (cont.)

Instruction	Semantics
add RD RS1 RS2	RD $:=$ RS1 $+$ RS2
sub RD RS1 RS2	RD $:=$ RS1 $-$ RS2
and RD RS1 RS2	RD $:=$ $\textsc{and}$(RS1, RS2)
or RD RS1 RS2	RD $:=$ $\textsc{or}$(RS1, RS2)
xor RD RS1 RS2	RD $:=$ $\textsc{xor}$(RS1, RS2)

Example

The assembly instruction add 4 8 12 means store in $\mathrm{R}4$ the sum of the contents of registers $\mathrm{R}8$ and $\mathrm{R}12$. The assembly instruction or 1 2 3 means store in $\mathrm{R}1$ the bitwise $\textsc{or}$ the contents of registers $\mathrm{R}2$ and $\mathrm{R}3$. Hence, if $\mathrm{R}2 = (01)^{16}$ and $\mathrm{R}3 = (10)^{16}$, then $\mathrm{R}1 \leftarrow 1^{32}$.

Test Instructions (I-type)

Instruction	Semantics
`s`*rel*`i RD RS1 imm`	RD := 1, if condition is satisfied,
	RD := 0 otherwise
if *rel* =`lt`	test if RS1 $<$ sext(imm)
if *rel* =`eq`	test if RS1 $=$ sext(imm)
if *rel* =`gt`	test if RS1 $>$ sext(imm)
if *rel* =`le`	test if RS1 $\leq$ sext(imm)
if *rel* =`ge`	test if RS1 $\geq$ sext(imm)
if *rel* =`ne`	test if RS1 $\neq$ sext(imm)

Example

The assembly instruction slti 4 8 -12 means store in $\mathrm{R}4$ the value '1' if $[\mathrm{R}8] < -12$, otherwise store in $\mathrm{R}4$ the value '0'. The assembly instruction snei 1 8 9 means store in $\mathrm{R}1$ the value '1' if $[\mathrm{R}8] \neq 9$, otherwise store in $\mathrm{R}1$ the value '0'.

Branch/Jump Instructions (I-type)

Instruction	Semantics
beqz RS1 imm	PC $=$ PC $+ 1 +$ sext(imm), if RS1 $= 0$
	PC $=$ PC $+ 1$, if RS1 $\neq 0$
bnez RS1 imm	PC $=$ PC $+ 1$, if RS1 $= 0$
	PC $=$ PC $+ 1 +$ sext(imm), if RS1 $\neq 0$
jr RS1	PC $=$ RS1
jalr RS1	R31 $=$ PC$+$1; PC $=$ RS1

Miscellaneous Instructions (I-type)

- The first special instruction is a the “no operation” (`special-nop`) instruction. This instruction has a null effect, and the only thing that happens during its execution is that the PC is incremented.
- The second special instruction is the “halt” (`halt`) instruction. This instruction causes the microprocessor to “freeze” and stop the execution of the program. Halting is implemented simply by not updating the PC.

Examples of Program Segments

First...RESAb3 review.

Examples of Program Segments: Example #1

Example

Convert the C code segment below to a simplified DLX's machine code.

```
    if (i==j)
      goto L1;
    f=g+h;
L1: f=f-i;
```

Example #1 (cont.)

Example

First, we assign a register to each of the variables in program segment. The register assignment appears in the following table.

Variable	Register
f	R1
g	R2
h	R3
i	R4
j	R5

Table: Register assignment.

Example #1 (cont.)

Example

Now, we convert the C code to a DLX's machine code. The conversion is depicted in the following table.

C code	DLX's machine code
if (i==j)	xor R6 R4 R5
goto L1;	beqz R6 1
f=g+h;	add R1 R2 R3
L1: f=f-i;	sub R1 R1 R4

Table: Conversion of the program segment in the example to the instruction set of the DLX.

Example#2

In the last example, every C instruction is mapped to a single machine code instruction. This is not always the case, as we are about to see in the following example.

Example

Convert the C code segment below to a simplified DLX's machine code.

```
   LOOP: g=g+A[i];
         i=i+j;
         if (i!=h) goto LOOP;
```

Example#2 (cont.)

Example

Again, we assign a register to each of the variables in program segment. The register assignment appears in the following table.

Variable	Register
g	R1
h	R2
i	R3
j	R4
A	R5
A+i	R6
A[i]	R7
i!=h	R8

Table: Register assignment. The variables below the line are temporary registers used for evaluating the program segment in the example.

Example#2 (cont.)

Example

Now, we convert the C code to a DLX's machine code. The conversion is depicted in the following table.

C code	DLX's machine code
LOOP: g=g+A[i];	add R6 R5 R3
	lw R7 R6 0
	add R1 R1 R7
i=i+j;	add R3 R3 R4
if (i!=h) goto LOOP;	xor R8 R3 R2
	bnez R8 -6

Table: Conversion of the program segment in the example to the instruction set of the DLX.