STUDENT GUIDE

SESSION 1: Introduction to DLX

Resources

- Documents “Essentials of DLX Instruction Set” and “The Myth of the variable” and Class slides.
- The DLX processor is described in detail in:
  - Computer Architecture. a quantitative approach "by Hennessy-Patterson, Mc Graw Hill, 1st Edition 1993. Chapters 5 and 6, Pages 172 (Instruction Set), and 273 onwards (pipeline characteristics), the reading of which is widely recommended.
- DLXVSim for Windows: an assembler and simulator of the DLX processor. Available in the virtual machine we use. If it is not previously installed, simply unzip DLXV3.1.zip into a folder and run the installation program leaving it in c: \ DLXV which is usually the default location. It is useful to define a shortcut from the Desktop. The "**HELP**" that includes the installation details the directives and other aspects of the tool and the assembler.

Method

In this first session we will get acquainted with the DLX instruction set, the processor with which we will study some aspects of pipelining. To work a little with the DLX, we will use a compilation (assembly) and simulation tool called DLXVSIM. We will describe its basic functions and the notation, directives and syntax of the DLX assembly language. Later we will use a second tool to analyze other aspects of segmentation and its risks.

Example program

This example shows the sum of the first n (10) integers. Write it and save it in a file named sumn.s The extension "s" identifies an assembly source.

```
        .data 100
n:      .word 10
sum: .space 4

   .text 1000
ini:   lw   r7,n(r0)      ; this is a COMMENT r7 <-- M(r0+n)=M(0+100)=10 r7 will
have a 10
        xor  r1,r1,r1
loop:    add      r1,r1,r7
        subi r7,r7,1
        bnez r7,loop
         nop
        sw   sum(r0),r1
   trap #6
```

.data and **.text** are directives of the assembler to locate the segments of data (variables) and text (program) in those directions. See the HELP of the tool

.word and **.space** are also assembly directives, such as **.float** or **.double**:

- **.word** is used to assign a logical name (**n**) to address 100 (in this case since it is the first variable in the data segment) where a space of a word with the value 10 has been reserved.
- **.space** is used to reserve bytes. In the example, 4 bytes from the sum address, which in this case is 104 since a word has 4 bytes.

ini: and **loop:** are labels that make the reference to addresses 1000 and 1008 more comfortable (the instructions measure all 4 bytes in this machine).

Semicolon denotes **comment**.

trap #6 causes the execution to be stopped but the instructions that entered the pipeline are completed. The rest is the repertoire itself, you must make sure you interpret and understand it. We will use this program to demonstrate the use of DLXVSim.

DLXVSIM

The teacher will describe the tool. When executing step by step DLXVSim first executes a sentence and stops before executing the next one, showing the latter. Consequently the first sentence of the program **is not seen**.

Once installed, create a program with the editor (UltraEdit), that is, a text file with the extension ".s" in which you will write the code of the sample program provided. Make sure you can:

- Assemble it in DLXVSim,
- Run it step by step,
- Test the registers,
- Test the data memory (check the result written in memory)
- See statistics.

1. See the document "**Essentials of DLX Instruction Set**" and look at the **tricks** provided at the end. Read this document carefully and also "**The Myth of the variable**".
2. Modify the program to make the sum of **the first n EVEN numbers** (not up to **n** but the **n** first, ie. the first 9 even numbers are 2 4 6 8 10 12 14 16 and 18 and not 2,4,6 and 8 which are the ones before 9). You **must** use the SLLI shift instruction (see documentation).
3. AT HOME: modify it again to make the sum of the first n odd numbers.
4. Implement the following algorithm, execute it and check the result. Define a data area that starts at address 100 that has an integer "nterm" of value 8. Reserve next space for 8 integers by labelling "fibo" to address 104. Develop the program from address 2000. Run it by showing in the data window the entire data area up to 2 integers beyond the last one that writes the program.

```
R1 <-- R0+R0                bucle:
R2<-- R0+1                     R3<-- R1+R2          R5<-- R5+4
R3<-- R0+1                     M(fib+R5)<-- R3      R4<-- R4-1
R4<-- M(R0+nterm)              R1<-- R2             if (R4 != 0) GOTO bucle
R5<-- R5 xor R5                R2<-- R3
```

5. Define a data area with two lists A and B of 12 integer numbers (N = 12 must be another variable) and make a program that adds them element by element and saves them in another list C holding the 12 results. For that list C, space must be reserved in the data area with the ".space" directive.

   Use two registers as loop counter and index, similar to the role of R4 and R5 in the previous exercise.

   Develop two variants of the program:

   a) The first one runs through elements in increasing order (register index 0,4,8,..4(N-1))

   b) The second variant runs through elements in decreasing order (register index 4(N-1), 4(N-2), ...4, 0)

   Execute it step by step, observing how results are produced in the data area. Also, observe the "Statistics" with cycles consumed.

6. Rewrite the following algorithm using the DLX Instruction Set giving appropriate values to the desired input variables (for example, lists of n = 6 elements and define the list of numbers similar to the previous exercise):

```
k=5;
for (i=0;i<n;i++)
{
   c[i] = k + b[i];
   k = k + a[i];
}
```

   You will need registers for:

- LOOP COUNTER (load with number of elements held in n=variable in memory)
- INDEX of lists for memory access. Increasing values from 0 or decreasing (incr./decr. ± 4)
- ACCUMULATOR for k
- Load a[i] and b[i], and then operate with its values, storing results in c[i]

7. Write and execute the following program changing the value for **n** (declare it**)**. Observe the values for registers. Answer the questions posed aside for each value of n used, learning how to count cycles consumed.

`lw r7,n(r0)`
`addi r1,r0,n`
`sub r2,r2,r2`
`loop:`
`subi r7,r7,1`
`add r3,r1,r2`
`nop`
`add r4,r1,r2`
`bnez r7,loop`
`nop`
`nop`
`trap #6`
	Instructions executed before the loop_____
	Instructions executed in the loop_____
	Instructions executed after the loop_____
	Total instructions_____ Total cycles_____
	CPI= _____
	Check Total cycles in Statistics (DLXVSim)_____

8. Write and execute the following program changing the value for **n** and **a(i)**. Include zero and non-zero values for **a**. Declare all needed variables. Answer the questions posed aside for each value of n used.

```
    lw   r7,n(r0)
    xor   r2,r2,r2
    xor   r3,r3,r3
loop:
   lw        r1,a(r2)
   subi      r7,r7,1
   addi      r2,r2,4
   beqz      r1,xxx
   nop
   add       r3,r3,r1
xxx:
   bnez     r7,loop
   nop
   sw nzval(r0),r3
  trap #6
```

a) What does the program do?

b) Zero values in a___

c) Non-zero values in a____

d) Instructions executed before the loop_____

e) Instructions executed in the loop_____

f) Instructions executed after the loop_____

g) Total instructions_____ Total cycles_____

h) CPI= _____

i) Check Total cycles in Statistics (DLXVSim)____

The first branch instruction in this later program implements an “if” instruction whereas the second branch implements a loop ¿Why, what is the difference?