Investigating Cache Technology

Introduction

Objectives

At the end of this lab you should be able to:

- Use the simulator to investigate different cache configurations
- Demonstrate the effectiveness of a cache for different programs
- Plot the miss rate as the function of cache size

Processor (CPU) Simulators

The computer architecture tutorials are supported by simulators, which are created to underpin theoretical concepts normally covered during the lectures. The simulators provide visual and animated representation of mechanisms involved and enable the students to observe the hidden inner workings of systems, which would be difficult or impossible to do otherwise. The added advantage of using simulators is that they allow the students to experiment and explore different technological aspects of systems without having to install and configure the real systems.

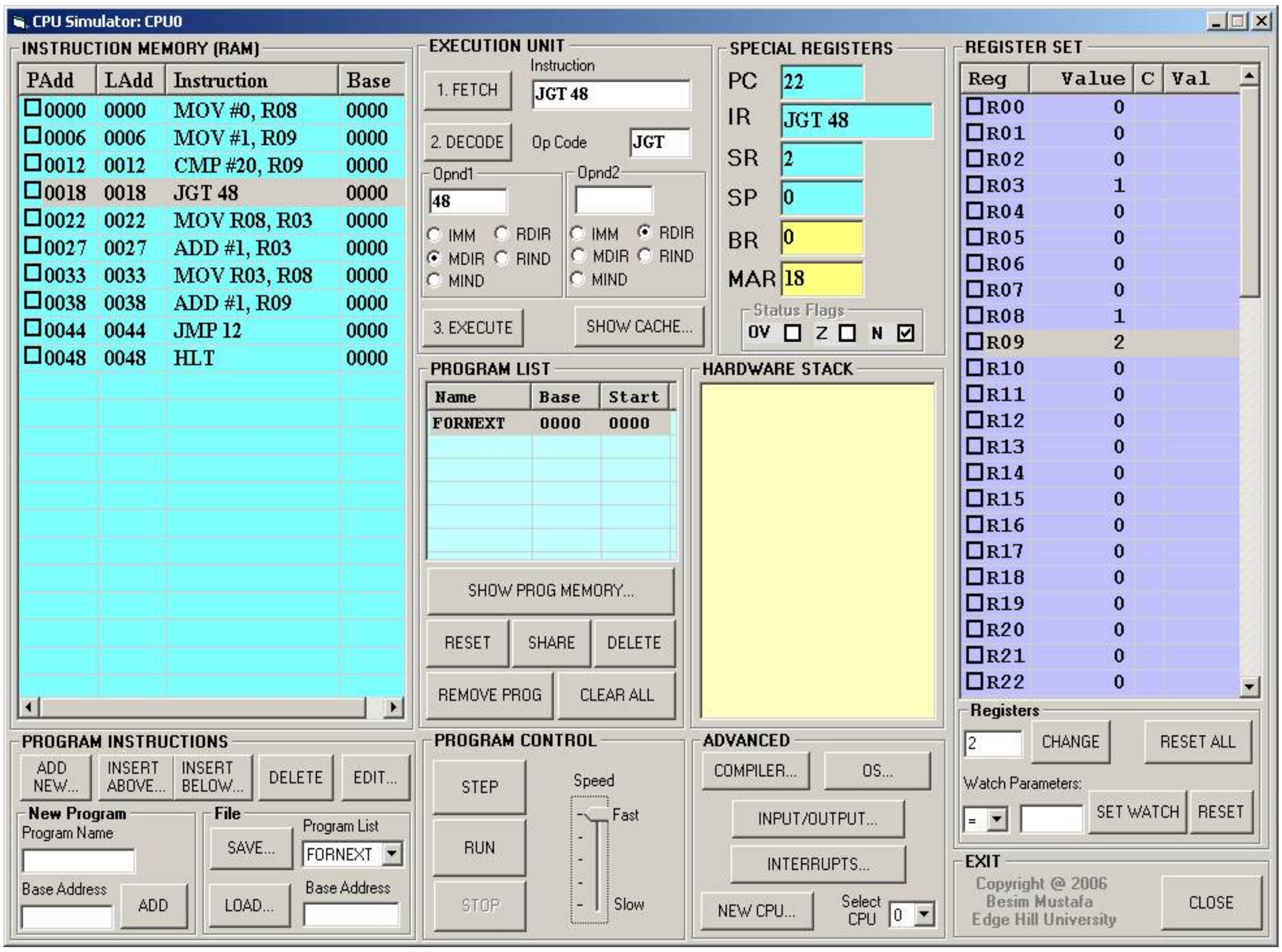
Basic Theory

Cache memories are used to improve performances of modern CPUs. They are placed between the RAM and the CPU and act as reservoirs of CPU instructions and data which can be fetched much faster than can be fetched directly from RAM. Different cache placement and replacement methods are being employed in industry.

Simulator Details

This section includes some basic information on the simulator, which should enable the students to use the simulator. The tutor(s) will be available to help anyone experiencing difficulty in using the simulator.

The simulator for this lab is an application running on a PC and is composed of multiple windows.



PAdd	LAdd	Instruction	Base
0000	0000	MOV #0, R08	0000
0006	0006	MOV #1, R09	0000
0012	0012	CMP #20, R09	0000
0018	0018	JGT 48	0000
0022	0022	MOV R08, R03	0000
0027	0027	ADD #1, R03	0000
0033	0033	MOV R03, R08	0000
0038	0038	ADD #1, R09	0000
0044	0044	JMP 12	0000
0048	0048	HLT	0000

Instruction: JGT 48
Op Code: JGT
Opnd1: 48 — IMM, RDIR, [✓ MDIR], RIND, MIND
Opnd2: — IMM, [✓ RDIR], MDIR, RIND, MIND

PC 22
IR JGT 48
SR 2
SP 0
BR 0
MAR 18
Status Flags: OV ☐ Z ☐ N ☑

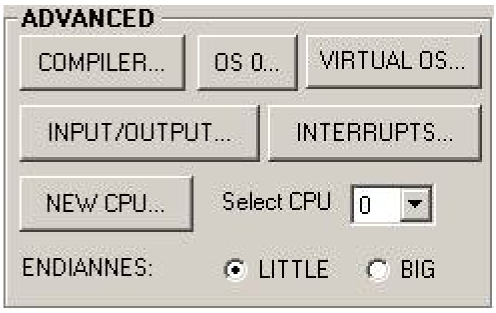
Name	Base	Start
FORNEXT	0000	0000

Reg	Value	C	Val
R00	0		
R01	0		
R02	0		
R03	1		
R04	0		
R05	0		
R06	0		
R07	0		
R08	1		
R09	2		
R10	0		
R11	0		
R12	0		
R13	0		
R14	0		
R15	0		
R16	0		
R17	0		
R18	0		
R19	0		
R20	0		
R21	0		
R22	0		

Registers: 2
Watch Parameters: =
Program List: FORNEXT
Select CPU: 0

Image 1 - Main simulator window

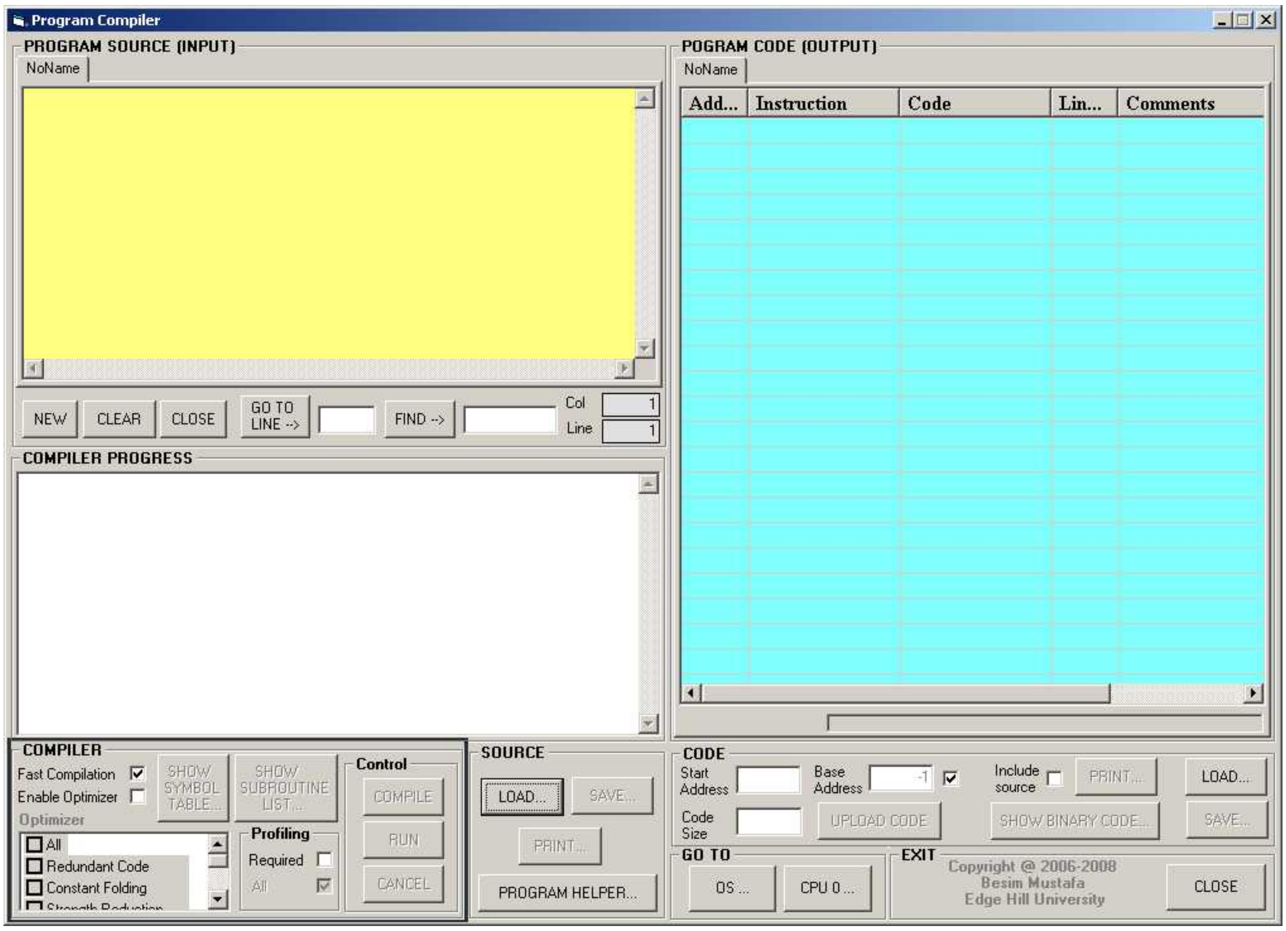
The main window shown in Image 1 is composed of several sub-views, which represent different functional parts of the simulated processor. For this lab session we are interested only in the compiler part of the simulator.



Select CPU: 0
ENDIANNES: [✓ LITTLE] BIG

Image 2 - Advanced functions

In order to access the compiler, click on the **COMPILER…** button as shown in Image 2 on the right. The compiler window shown in Image 3 below will show.



Fast Compilation ✓
Enable Optimizer ☐
Required ☐
All ✓
Col 1
Line 1
Base Address -1 ✓
Include source ☐

Image 3 - The main compiler window

In the compiler window there are three main sub-windows

- **Program Source** - all high-level source statements appear here.
- **Compiler Progress** - information on the progress of a compilation appear here.
- **Program Code** - assembly code generated by the compiler appear here.

Lab Exercises - Investigate and Explore

The lab exercises are a series of experiments, which are attempted by the students under guidelines. The students are expected to carry out further investigations on their own in order to form a better understanding of the technology.

Now, have a go at the following activities:

1. Add a new program (use **ADD** in **PROGRAM INSTRUCTIONS** view in the CPU simulator window). Use a suitable name and the base address 0. Click on the **ADD NEW…** button and add the following instruction repeated 16 times

   ```
   MOV #0, R00
   ```

   …and finish off with the instruction

   ```
   HLT
   ```

   As you enter a new code or click on an existing one in the **INSTRUCTION MEMORY** view, the

   Click on the **SHOW PIPELINE…** button and check the **No instruction pipeline** check box. Close the **Instruction Pipeline** window.

   Click on the **SHOW CACHE…** button. Check the **Stay on top** check box in **Instruction Cache** window.

2. Refer to **Appendix A**. Select the appropriate cache parameters and verify the cache behaviour as suggested in this appendix. Make a note of your observations.
3. Refer to **Appendix B**. Select the appropriate cache parameters and verify the cache behaviour as suggested in this appendix. Make a note of your observations.
4. Refer to **Appendix C**. Select the appropriate cache parameters and verify the cache behaviour as suggested in this appendix. Make a note of your observations.
5. Enter the following code and compile

   ```
   program ctest1
       for n = 1 to 5
           for i = 1 to 5
               k = k + 2
               j = j + 1
           next
       next
   end
   ```

   Make a note of the code size.

   Load it in the instruction memory (use the **LOAD IN MEMORY** button). Show the **Instruction Cache** window and make sure it stays on top of all other windows. Next select the following cache parameters

   Type: Direct Mapped

Block size: 4

In the **Instruction Cache** window, check the **Enable chart** check box.

For each of the cache sizes 4, 8, 16 and 32 do the following

In the **CPU Simulator** window, click the **RESET** button; slide the speed control to the fastest position (at the top of the slide); click the **RUN** button and observe the chart. When the program stops make a note of the value of the bar against the currently selected cache size.

6. Enter the following code and compile

```
program ctest2
    sub p1
        x = x + 1
    end sub

    sub p2
        x = x + 2
    end sub

    for n = 1 to 10
        call p1
        call p2
    next
end
```

Make a note of the code size and compare it against the one from (5).

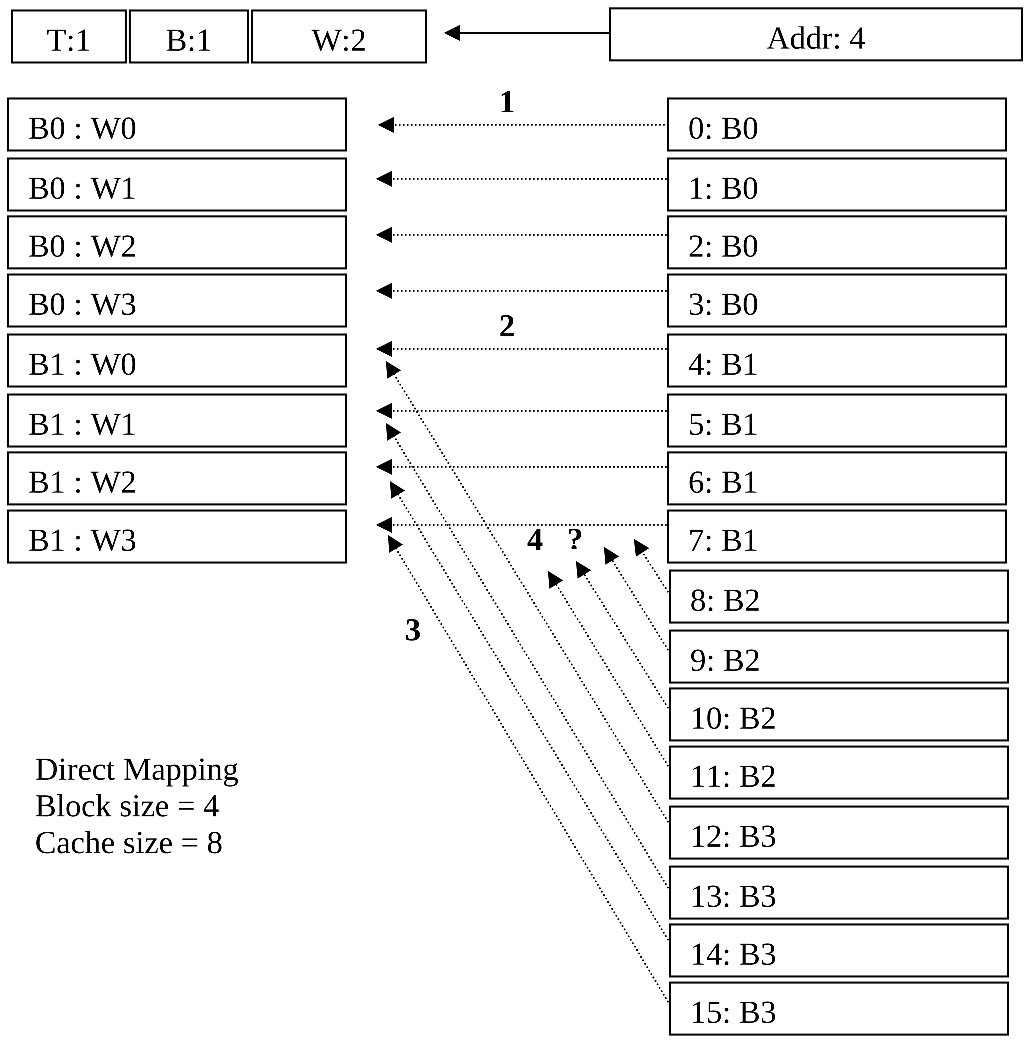
Follow the same instructions as in (5) above and make a note of the values of the bars against the respective cache sizes.

7. Compare the results of (5) and (6) above and comment on any differences. Make a note of an explanation for any differences. Clue: You need to consider the way the two programs are written and work.

Appendix A

Direct Mapping

Cache: 8 words, 4-word block **RAM**: 16 words, 4 blocks

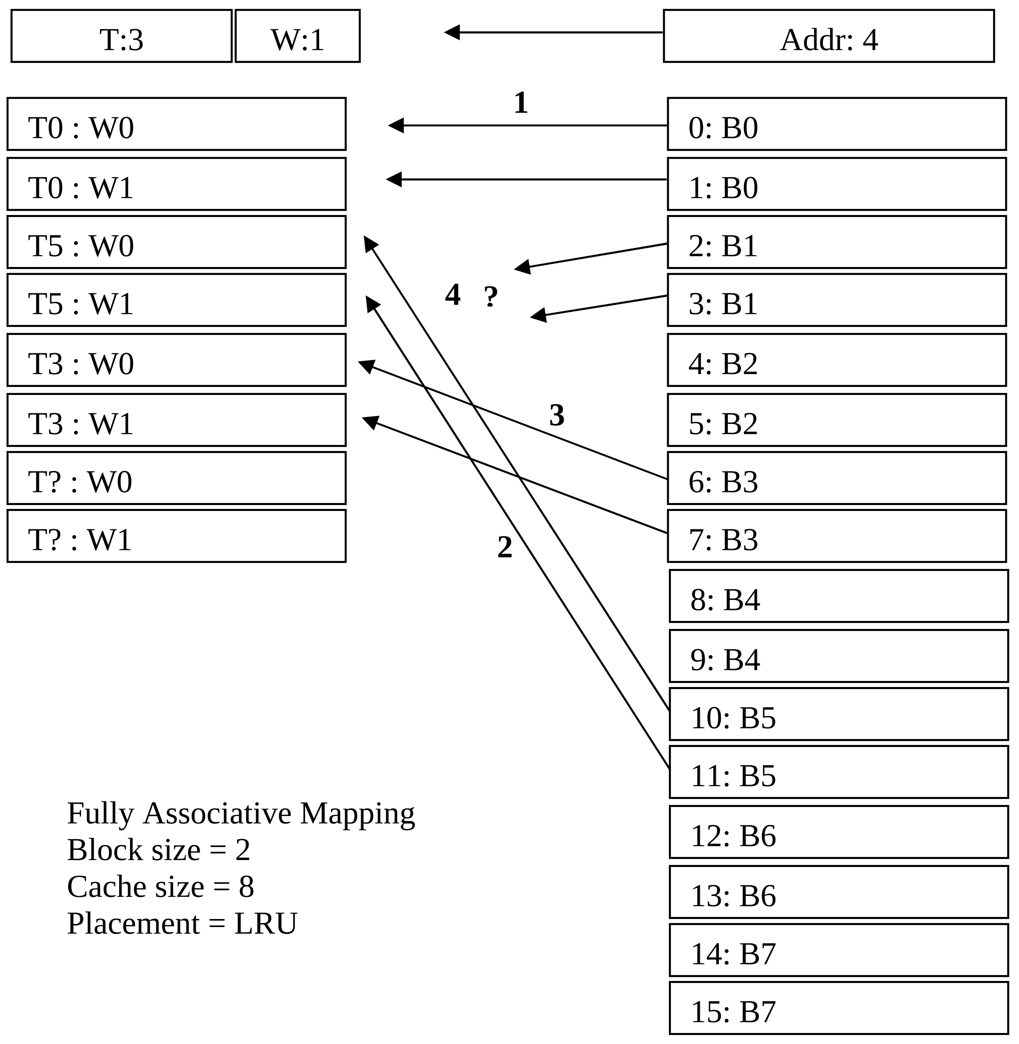


Direct Mapping
Block size = 4
Cache size = 8

Appendix B

Associative Mapping

Cache: 8 words, 2-word block **RAM**: 16 words, 8 blocks

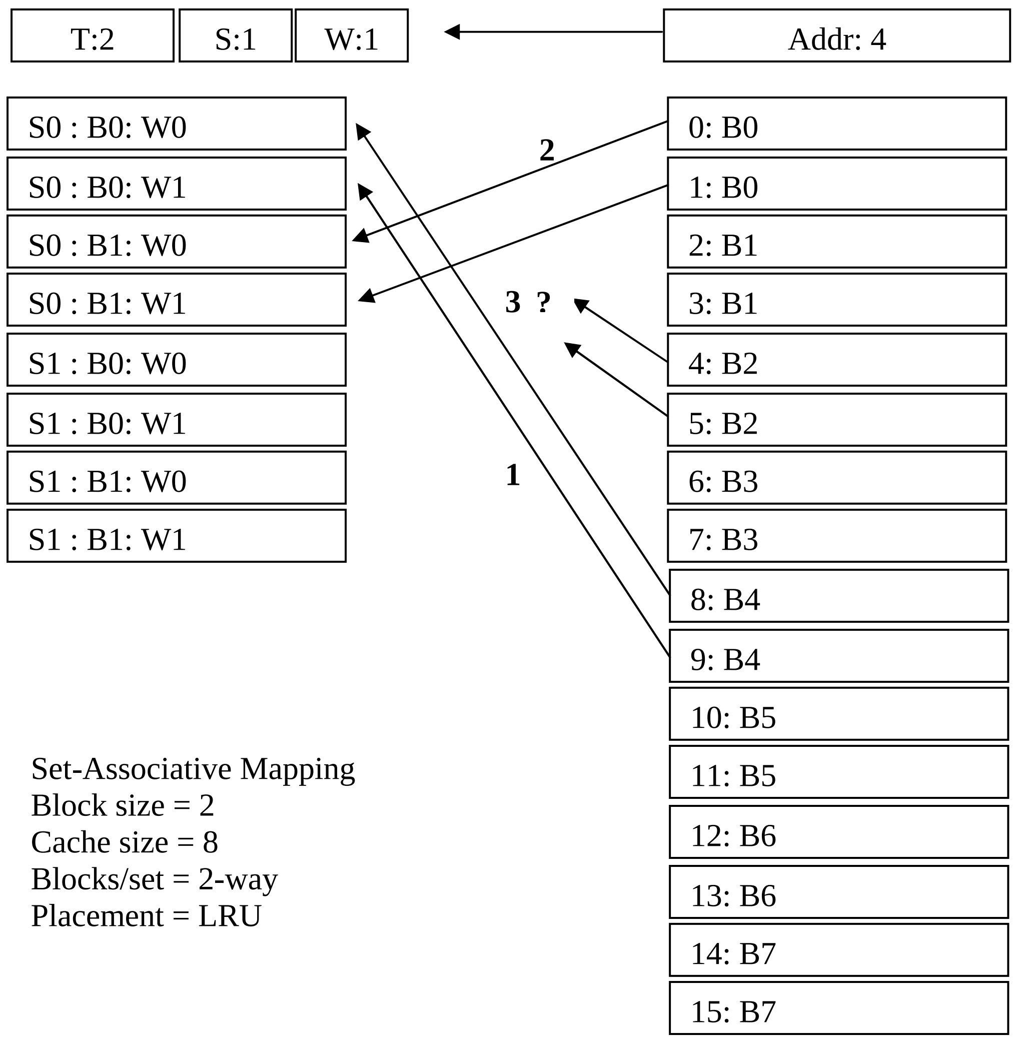


Appendix C

Set-Associative Mapping

Cache: 8 words, 2-word block,
2-block set, i.e. 2-way

RAM: 16 words, 8 blocks



Set-Associative Mapping
Block size = 2
Cache size = 8
Blocks/set = 2-way
Placement = LRU