**MPCA Lab Week 9: Working on Pipeline and Cache simulator**

**Task1: 5 stage pipeline Simulator.**

Link: [MIPS Five Stage Pipeline](http://www.ecs.umass.edu/ece/koren/architecture/windlx/main.html) (umass.edu)

Consider the following instructions. Execute these instructions using 5 stage pipeline - MIPS architecture simulator.

ADD R0, R1, R2

SUB R3, R0, R4

FP\_LOAD F1, Offset,R1

FP\_ADD F5,F1,F1

**Screenshot of 5stage pipeline simulator when above instructions are executed without data forwarding:**

Table

Description automatically generated

**Observe the following and note down the results. Check whether there is data dependency among the instructions?**

Yes, RAW hazard in instructions 0 and 1 for R1 as there is no data forwarding the value of R1 can be obtained only after WB stage. Another RAW hazard in instructions 2 and 3 for F1 and this cannot be eliminated even with data forwarding (but no. of stalls will reduce).

**If yes, then, how many stall states have been introduced?**

2 stalls in instruction 1 and 2 and 2 more stalls in instruction 3. So, totally **4** stalls.

**If data forwarding is applied how many stall states have been reduced?**

Table

Description automatically generated

The above screenshot is for the same with data forwarding. As we can see the number of stalls is reduced from 4 to 2.

**Mention the total number of clock cycles used with and without data forwarding.**

With Data Forwarding **9** Cycles

Without Data Forwarding **12** cycles.

**Task2: Cache Simulator:**

**Link:** [**ParaCache Main Menu (ntu.edu.sg)**](https://www3.ntu.edu.sg/home/smitha/ParaCache/Paracache/start.html)

1. A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte.

**Screenshot of Paracache Simulator:**

A picture containing graphical user interface

Description automatically generated

**(a) Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address using direct mapped Cache.**

Offset/Word = 64 = 26 = 6 bits  
Index bits/Block = log2(2048/64) = 5 bits  
Instruction Length = log2(65536) = 16 bits  
Tag = 16 bits - 6 bits - 5 bits = 5 bits

**(b) When a program is executed, the processor reads data sequentially from the following word addresses: 128, 144, 2176, 2180, 128, 2176 All the above addresses are shown in decimal values (Convert the address to hexadecimal equivalent of the decimals given above and submit in the simulator). Assume that the cache is initially empty. For each of the above addresses, indicate whether the cache access will result in a hit or a miss.**

A picture containing text, screenshot, indoor

Description automatically generated128=0x80

144=0x90

2176=0x880

2180=0x884

128=0x80

2176=0x880

|  |  |  |
| --- | --- | --- |
| **Decimal** | **Hexadecimal Equivalent** | **Hit/Miss** |
| 128 | 80 | Miss |
| 144 | 90 | Hit |
| 2176 | 880 | Miss |
| 2180 | 884 | Hit |
| 128 | 80 | Miss |
| 2176 | 880 | Miss |

1. **For the above-mentioned problem, calculate and execute for 4way set associativity and fully associative mapping technique. For each technique randomly generate ten addresses and indicate whether the cache access will result in a hit or a miss. Assume block replacement policy as random.**

**4-way Set Associative:**

**(a) Calculate the number of bits in each of the Tag, Set, and Word fields of the memory address.**

Table

Description automatically generated

Offset/Word = log2( 64) = log2( 26) = 6 bits  
Index bits/Set = log2(2048/64/4) = 3 bits  
Instruction Length = log2(65536) = 16 bits  
Tag = 16 bits - 6 bits - 3 bits = 7 bits

**Randomly generated instructions:**

1b9f,eb4c,92f1,f7d0,c4c6,5562,3777,8ad4,d424,da2c

A picture containing graphical user interface

Description automatically generated

A picture containing diagram

Description automatically generated

|  |  |
| --- | --- |
| **Instruction** | **Hit/Miss** |
| 1b9f | Miss |
| Eb4c | Miss |
| 92f1 | Miss |
| F7d0 | Miss |
| C4c6 | Miss |
| 5562 | Miss |
| 3777 | Miss |
| 8ad4 | Miss |
| D424 | Miss |
| Da2c | Miss |

**Fully Associative:**

A screenshot of a computer

Description automatically generated with low confidence

Offset/Word = 6 bits  
Instruction Length = log2(65536) = 16 bits  
Block = 16 bits - 6 bits = 10 bits

Randomly generated instructions: e89a, a4fe,7c67,61b1,2018,96ce,797,5e5d,ec05,6ada

Calendar

Description automatically generated with medium confidence

Table

Description automatically generated

|  |  |
| --- | --- |
| **Instruction** | **Hit/Miss** |
| E89a | Miss |
| A4fe | Miss |
| 7c67 | Miss |
| 61b1 | Miss |
| 2018 | Miss |
| 96ce | Miss |
| 797 | Miss |
| 5e5d | Miss |
| Ec05 | Miss |
| 6ada | Miss |

e89a, a4fe,7c67,61b1,2018,96ce,797,5e5d,ec05,6ada

**Task3: Use Arm Simulator Given a ‘c’ code convert it in its equivalent Arm Code and execute in ARM simulator**

1. x = (a + b) – c

**Code:**

.data

a:.word 32

b:.word 21

c:.word 21

x:.word

.text

ldr r0,=a

ldr r1,=b

ldr r2,=c

ldr r3,[r0]

ldr r4,[r1]

ldr r5,[r2]

ldr r6,=x

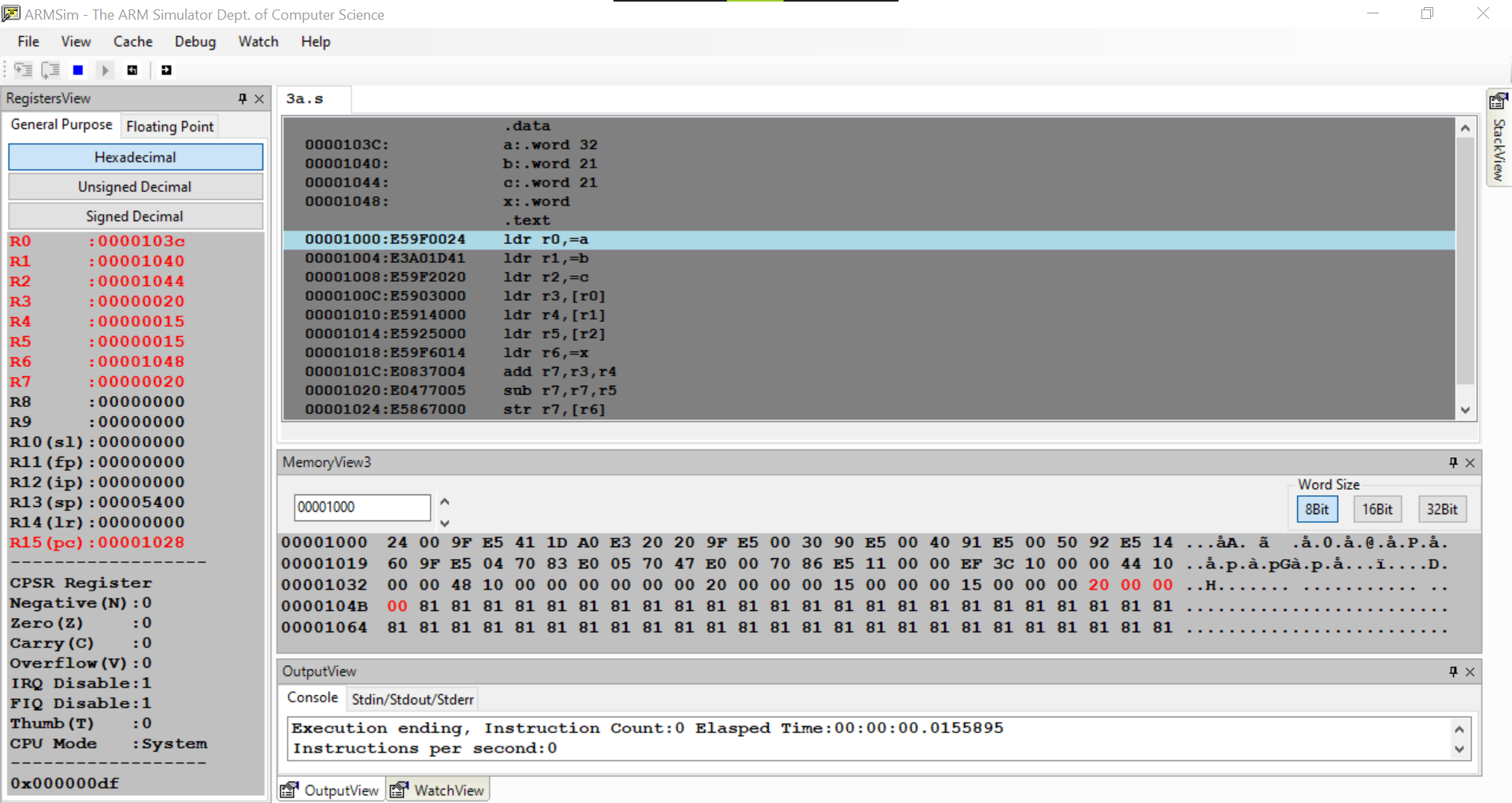
add r7,r3,r4

sub r7,r7,r5

str r7,[r6]

swi 0x11

**Screenshot:**

****

1. z = (a << 2) | (b & 15)

**Code:**

.data

a:.word 21

b:.word 12

z:.word

.text

ldr r0,=a

ldr r1,=b

ldr r2,=z

ldr r3,[r0]

ldr r4,[r1]

mov r5,r3,lsl #2

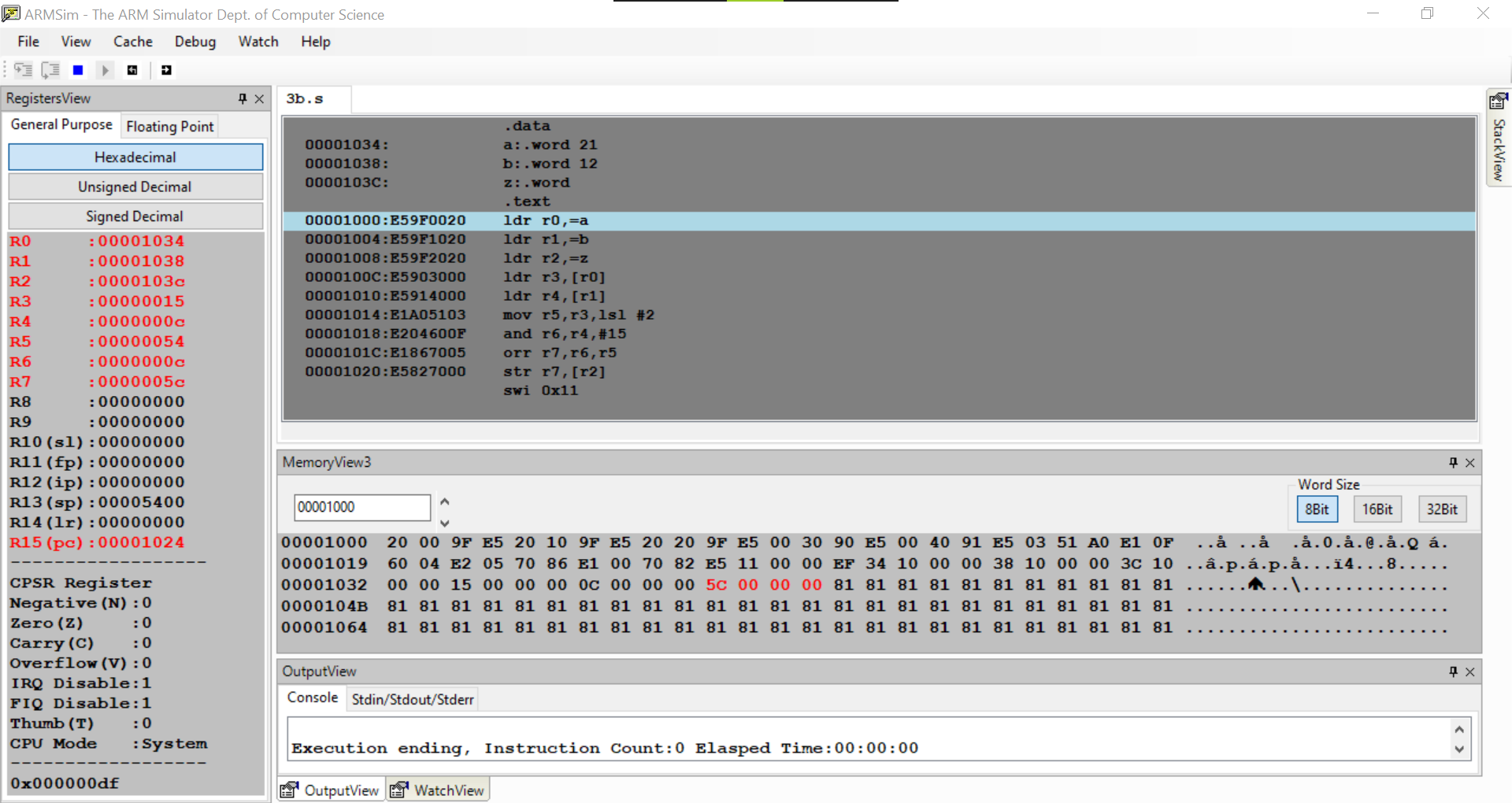
and r6,r4,#15

orr r7,r6,r5

str r7,[r2]

swi 0x11

**Screenshot:**

****