컴퓨터구조

Project #3

Pipeline Architecture

Class : **월 3 수 4**

Professor : **이성원 교수님**

Student ID : **2020202031**

Name : **김재현**

1. Introduction

In the pipelined architecture, there are several occasions when a necessary operation for an instruction cannot be performed because the requested resources or data are not available in the same pipeline stage at the same clock cycle. These events are called hazards, and there are three different types.

1. Structural hazards: The requested hardware resource is not ready to support the demanding instructions at the exact moment.

2. Data hazards: The result data of preceding instructions are going to be used for the coming instruction. But the result data are not delivered to the requesting pipeline stage at the exact moment.

3. Control hazards: The program is about to branch either way of two different flows. Then the following instructions from one of either direction should be not executed and should wait for the result of the branch instruction. The following figure shows an example of similar pipelined architectures from the textbook.

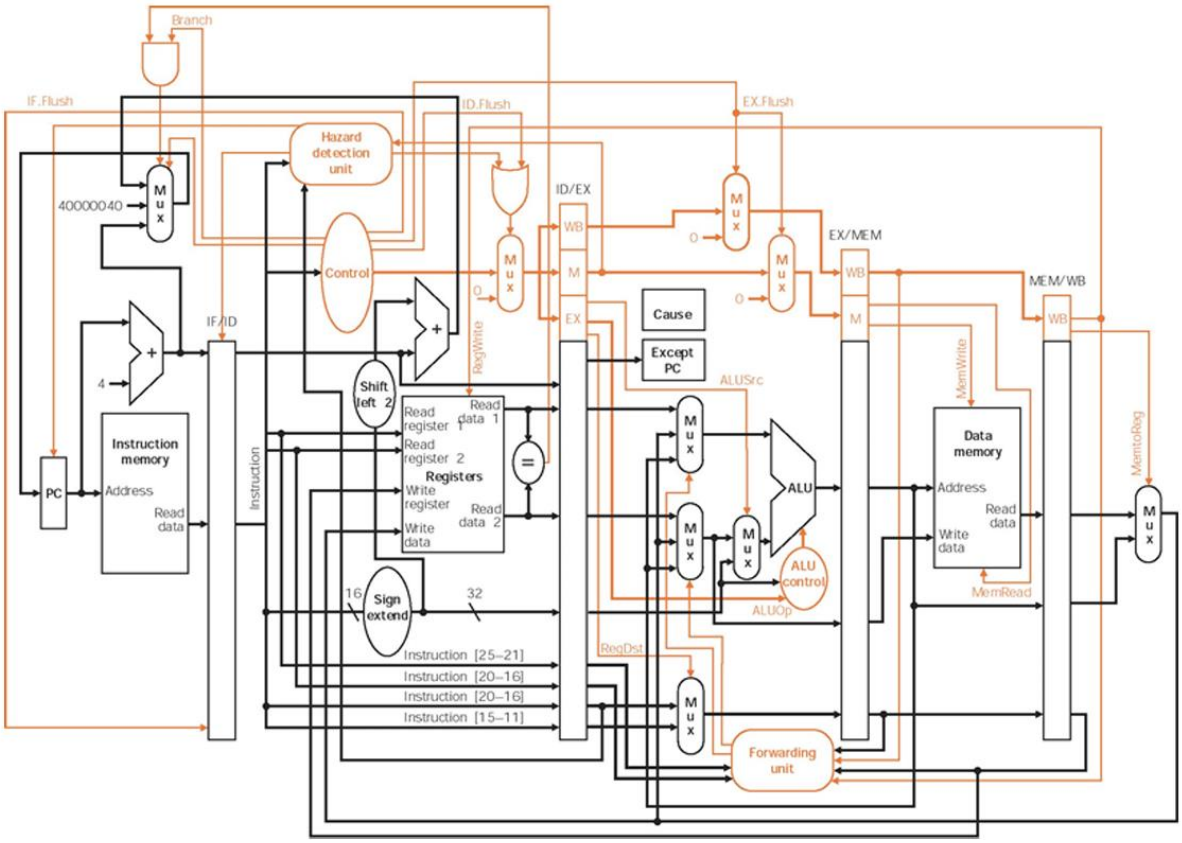


Figure 1 The example Pipeline CPU with hazards units

However, the pipelined processor without considering the hazards is given. In this assignment, modify the sorting program given by removing unnecessary NOPs and determine how to forward necessary data to avoid hazards. Simulate your code and discuss your results.

2.Assignment

기존 어셈블리코드에서 Forwarding 제어 없이 필요 없는 NOP 만 제거한 assembly code 는 radix_sort.asm 파일에,

기존 어셈블리코드에서 Forward 제어신호를 추가하여 더 많은 NOP 를 제거하여, 재구성된 assembly code 는 radix_sort_fwd.asm 파일에 각각 저장하므로써 두 코드를 구분했습니다.

Forwarding 제어 없이 작성한 assembly code 는 forward 신호가 전부 다 0 이기 때문에, init 을 위한 포워딩 신호파일인 radix_sort_fwd.txt 에 Forward 제어신호를 추가한 assembly code 포워드 신호를 저장했습니다.

결론:

no forwarding version assembly code -> radix_sort.asm

forwarding version assembly code -> radix_sort_fwd.asm

forwarding version forward signal -> radix_sort_fwd.txt

아래장부터 보고서 시작합니다.

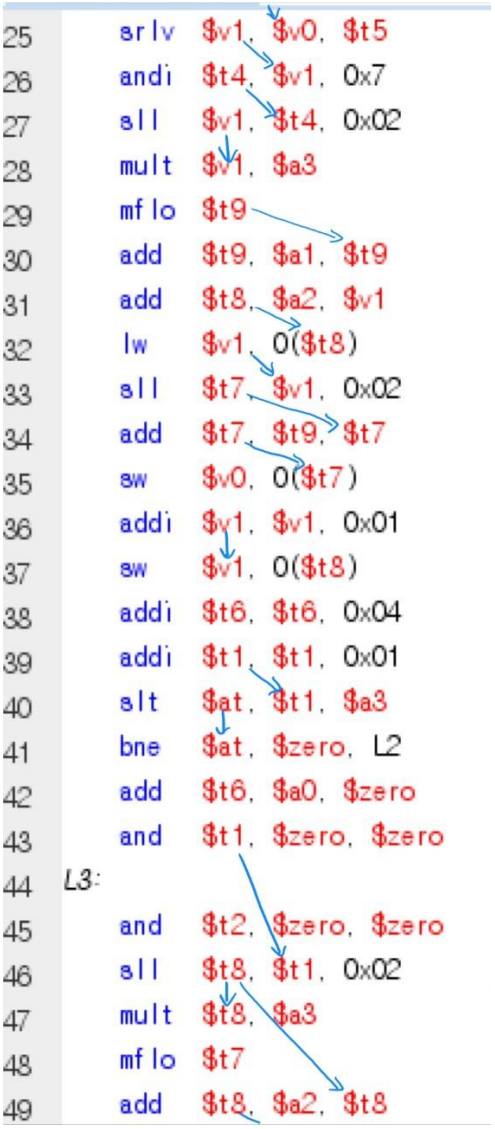
show data dependency

```
1     .text
2     .globl main
3
4  main:
5     ori   $a0, $zero, 0x2000
6     ori   $a1, $zero, 0x2280
7     ori   $a2, $zero, 0x2200
8     ori   $a3, $zero, 0x0010
9     ori   $s0, $zero, 0x0005
10    and   $t5, $zero, $zero
11    and   $t0, $zero, $zero
12 L0:
13    and   $t1, $zero, $zero
14    add   $t8, $a2, $zero
15 L1:
16    sw    $zero, 0($t8)
17    addi  $t8, $t8, 0x04
18    addi  $t1, $t1, 0x01
19    slti  $at, $t1, 0x08
20    bne   $at, $zero, L1
21    add   $t6, $a0, $zero
22    and   $t1, $zero, $zero
23 L2:
24    lw    $v0, 0($t6)
```

add 의 WR 단계에서 t8 에 값이 저장되는데 sw 가 EX 단계에서 이를 주소값 계산에 사용합니다. 따라서 data dependency 가 존재하며, 이를 해결하기 위해 add 와 sw 사이에 nop stall 을 3 개 삽입해줌으로써 add 의 WB 가 sw 의 ID 가 겹치지 않도록 해줍니다. add 와 addi 사이엔 4 개의 연산이 존재하므로, 자연스럽게 data dependency 가 사라지는 것을 확인할 수 있습니다.

같은 원리로 addi 와 slti 사이, slti 와 bne 사이에 각각 nop stall 3 개를 삽입함으로써 data dependency 를 해결할 수 있습니다.

add 와 lw 사이에도 data dependency 가 존재합니다. 하지만 이미 두 연산과 관계없는 and 연산이 존재하므로 and 연산 밑에 nop stall 을 2 개만 삽입하면 add 와 lw 사이에 3 개의 공간이 생기면서 data dependency 를 해결할 수 있습니다.



(lw, srlv), (srlv, andi), (andi, sll), (sll, mult), (mflo, add), (add, lw), (lw, sll), (sll, add), (add, sw), (addi, sw), (addi, slt), (slt, bne)사이에 각각 nop stall 3 개를 삽입함으로써 data dependency 를 해결할 수 있습니다.

and 와 sll 사이의 data dependency 를 해결하기 위해서는 두 명령어 사이에 두 명령어와 관련 없는 연산이 3 회 존재해야 합니다. 이미 and 명령어가 한 개 존재하므로, nop stall 2 개를 삽입함으로써 data dependency 를 해결할 수 있습니다.

sll 과 mult 사이, sll 과 add 사이에 data dependency 는 sll 과 mult 사이에 nop stall 3 개를 삽입함으로써 동시에 해결할 수 있습니다.

(add, lw), (lw, beq), (add, lw), (lw, sw), (addi, slt), (slt, bne), (add, slti), (slti, bne), (add, slt), (slt, bne) 사이에 각각 nop stall 3 개를 삽입함으로써 data dependency 를 해결할 수 있습니다.

solve data dependency by inserting stall

without data forwarding

```
1    .text
2    .globl main
3
4  main:
5    nop
6    ori   $a0, $zero, 0x2000
7    ori   $a1, $zero, 0x2280
8    ori   $a2, $zero, 0x2200
9    ori   $a3, $zero, 0x0010
10   ori   $s0, $zero, 0x0005
11   and   $t5, $zero, $zero
12   and   $t0, $zero, $zero
13 L0:
14   and   $t1, $zero, $zero
15   add   $t8, $a2, $zero
16   nop
17   nop
18   nop
19 L1:
20   sw    $zero, 0($t8)
21   addi  $t8, $t8, 0x04
22   addi  $t1, $t1, 0x01
23   nop
24   nop
25   nop
```

```
26   slti  $at, $t1, 0x08
27   nop
28   nop
29   nop
30   bne   $at, $zero, L1
31   nop
32   add   $t6, $a0, $zero
33   and   $t1, $zero, $zero
34   nop
35   nop
36 L2:
37   lw    $v0, 0($t6)
38   nop
39   nop
40   nop
41   srlv  $v1, $v0, $t5
42   nop
43   nop
44   nop
45   andi  $t4, $v1, 0x7
46   nop
47   nop
48   nop
49   sll   $v1, $t4, 0x02
```

```
50   nop
51   nop
52   nop
53   mult  $v1, $a3
54   mflo  $t9
55   nop
56   nop
57   nop
58   add   $t9, $a1, $t9
59   add   $t8, $a2, $v1
60   nop
61   nop
62   nop
63   lw    $v1, 0($t8)
64   nop
65   nop
66   nop
67   sll   $t7, $v1, 0x02
68   nop
69   nop
70   nop
71   add   $t7, $t9, $t7
72   nop
73   nop
```

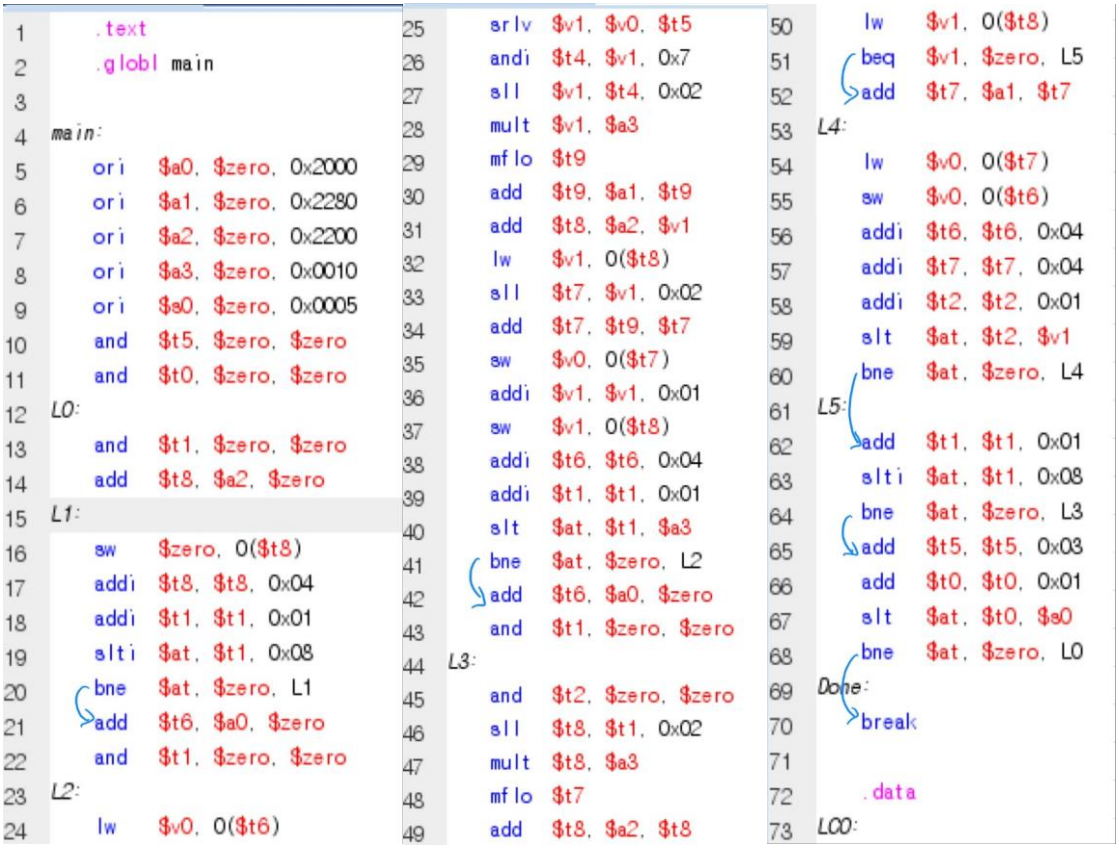
```
74   nop
75   sw    $v0, 0($t7)
76   addi  $v1, $v1, 0x01
77   nop
78   nop
79   nop
80   sw    $v1, 0($t8)
81   addi  $t6, $t6, 0x04
82   addi  $t1, $t1, 0x01
83   nop
84   nop
85   nop
86   slt   $at, $t1, $a3
87   nop
88   nop
89   nop
90   bne   $at, $zero, L2
91   nop
92   add   $t6, $a0, $zero
93   and   $t1, $zero, $zero
94 L3:
95   and   $t2, $zero, $zero
96   nop
97   nop
```

```
98   sll   $t8, $t1, 0x02
99   nop
100  nop
101  nop
102  mult  $t8, $a3
103  mflo  $t7
104  add   $t8, $a2, $t8
105  nop
106  nop
107  nop
108  lw    $v1, 0($t8)
109  nop
110  nop
111  nop
112  beq   $v1, $zero, L5
113  nop
114  add   $t7, $a1, $t7
115  nop
116  nop
117  nop
118 L4:
119  lw    $v0, 0($t7)
120  nop
121  nop
```

```
122  nop
123  sw    $v0, 0($t6)
124  addi  $t6, $t6, 0x04
125  addi  $t7, $t7, 0x04
126  addi  $t2, $t2, 0x01
127  nop
128  nop
129  nop
130  slt   $at, $t2, $v1
131  nop
132  nop
133  nop
134  bne   $at, $zero, L4
135  nop
136 L5:
137  add   $t1, $t1, 0x01
138  nop
139  nop
140  nop
141  slti  $at, $t1, 0x08
142  nop
143  nop
144  nop
```

```
145     bne   $at, $zero, L3
146     nop
147     add   $t5, $t5, 0x03
148     add   $t0, $t0, 0x01
149     nop
150     nop
151     nop
152     slt   $at, $t0, $s0
153     nop
154     nop
155     nop
156     bne   $at, $zero, L0
157     nop
158 Done:
159     break
```

show control dependency



```
1   .text
2   .globl main
3
4   main:
5       ori   $a0, $zero, 0x2000
6       ori   $a1, $zero, 0x2280
7       ori   $a2, $zero, 0x2200
8       ori   $a3, $zero, 0x0010
9       ori   $s0, $zero, 0x0005
10      and   $t5, $zero, $zero
11      and   $t0, $zero, $zero
12  L0:
13      and   $t1, $zero, $zero
14      add   $t8, $a2, $zero
15  L1:
16      sw    $zero, 0($t8)
17      addi  $t8, $t8, 0x04
18      addi  $t1, $t1, 0x01
19      slti  $at, $t1, 0x08
20      bne   $at, $zero, L1
21      add   $t6, $a0, $zero
22      and   $t1, $zero, $zero
23  L2:
24      lw    $v0, 0($t6)
25      srlv  $v1, $v0, $t5
26      andi  $t4, $v1, 0x7
27      sll   $v1, $t4, 0x02
28      mult  $v1, $a3
29      mflo  $t9
30      add   $t9, $a1, $t9
31      add   $t8, $a2, $v1
32      lw    $v1, 0($t8)
33      sll   $t7, $v1, 0x02
34      add   $t7, $t9, $t7
35      sw    $v0, 0($t7)
36      addi  $v1, $v1, 0x01
37      sw    $v1, 0($t8)
38      addi  $t6, $t6, 0x04
39      addi  $t1, $t1, 0x01
40      slt   $at, $t1, $a3
41      bne   $at, $zero, L2
42      add   $t6, $a0, $zero
43      and   $t1, $zero, $zero
44  L3:
45      and   $t2, $zero, $zero
46      sll   $t8, $t1, 0x02
47      mult  $t8, $a3
48      mflo  $t7
49      add   $t8, $a2, $t8
50      lw    $v1, 0($t8)
51      beq   $v1, $zero, L5
52      add   $t7, $a1, $t7
53  L4:
54      lw    $v0, 0($t7)
55      sw    $v0, 0($t6)
56      addi  $t6, $t6, 0x04
57      addi  $t7, $t7, 0x04
58      addi  $t2, $t2, 0x01
59      slt   $at, $t2, $v1
60      bne   $at, $zero, L4
61  L5:
62      add   $t1, $t1, 0x01
63      slti  $at, $t1, 0x08
64      bne   $at, $zero, L3
65      add   $t5, $t5, 0x03
66      add   $t0, $t0, 0x01
67      slt   $at, $t0, $s0
68      bne   $at, $zero, L0
69  Done:
70      break
71
72      .data
73  L00:
```

branch 명령어는 branch 조건에 대한 판단을 ID stage 에서 진행하므로, branch 명령어가 ID stage 로 넘어갈 때, 다음 명령어가 IF stage 에 같이 들어오게 됩니다. 만일 branch 조건이 부합하여 다른 label 로 jump 하게 되면, IF 에 들어와있던 명령어는 예상치 못한 결과를 도출할 수 있으므로, IF 에 들어가게 될 명령어가 아무런 역할도 하지 못하게 해야합니다. 따라서 branch 명령어 바로 뒤에 무조건 아무런 역할도 수행하지 않는 nop stall 을 한 개 삽입해야 합니다. 이로써 Control Dependency 를 해결할 수 있습니다.

data forwarding to ALU

실제 프로세서서는 동적으로 Hazard 를 확인하기 때문에 문제가 없으나, 정적으로 Forwarding 방향을 지정하는 본 시뮬레이션에서는 Control dependency 에 따라 해결할 수 없는 data hazard 문제가 발생하게 됩니다.
우리는 명령어 간의 data dependency 를 해결하기 위해 data forwarding 을 줍니다. 하지만 본 시뮬레이션은 data forwarding 을 정적으로 지정해야 하는 한계 때문에, 각 Label 의 상단에 위치한 명령어들이 이전 Label 에 속한 명령어들과 data dependency 가 존재한다면, Label 로 들어갈 때, 이전 Label 에서 그대로 내려온 것인지, branch 명령어의 jump 조건을 만족하여 jump 한 것인지 구분할 수 없는 문제가 발생합니다.

하지만 본 시뮬레이션에서는 이전 주소의 포워딩 신호가 다음 주소의 명령어에 적용되므로 각 Label 의 첫 번째 명령어는 이전 Label 과 data dependency 가 존재해도 상관 없고, 첫 번째 명령어를 제외한 나머지 명령어는 현재 Label 내에서만 data dependency 가 존재해야 합니다.

L0 의 and 와 L1 의 addi 는 서로 다른 Label 에 존재하므로 이 둘 사이의 dependency 는 존재하면 안됩니다. 따라서 nop stall 2 개를 삽입해줘야 합니다. 그러나 nop stall 2 개를 삽입하면 L0 and 와 L1 sw 사이에 data dependency 를 해결할 수 없으므로, total nop stall 3 개를 삽입합니다.

L1 의 addi 와 slti 는 data forwarding 을 통해 nop 를 줄여줄 수 있습니다.

bne 는 $t, $s 를 사용한 계산을 ID 단계에서 진행하기 때문에 ALU 로 forwarding 을 진행하는 본 시뮬레이션에서는 data dependency 를 해결할 수 없으므로, 무조건 nop 가 3 개 삽입돼야 합니다.

L1 add 와 L2 lw 사이의 data dependency 는 WB->ALU 로 해결할 수 있습니다.

lw 는 WB stage 에서 비로소 register 에 값을 write 하므로, WB->ALU forwarding 만이 가능합니다. 따라서 lw 가 WB stage 로 갈 때, MM stage 에는 nop 가 들어가야합니다. 따라서 lw 명령어 다음에 data dependency 가 존재하는 명령어가 온다면, lw 다음에는 nop 1 개가 삽입돼야 합니다.

L2 lw 와 L2 srlv 사이의 data dependency 는 WB->ALU 로 해결할 수 있습니다.

L2 srlv 와 L2 andi 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 andi 와 L2 sll 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 sll 와 L2 mult 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 mflo 와 L2 add 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 add 와 L2 lw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 lw 와 L2 sll 사이의 data dependency 는 WB->ALU 로 해결할 수 있습니다.

L2 sll 와 L2 add 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 add 와 L2 sw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 addi 와 L2 sw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L2 addi 와 L2 slt 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L3 sll 와 L3 mult 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L3 add 와 L3 lw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

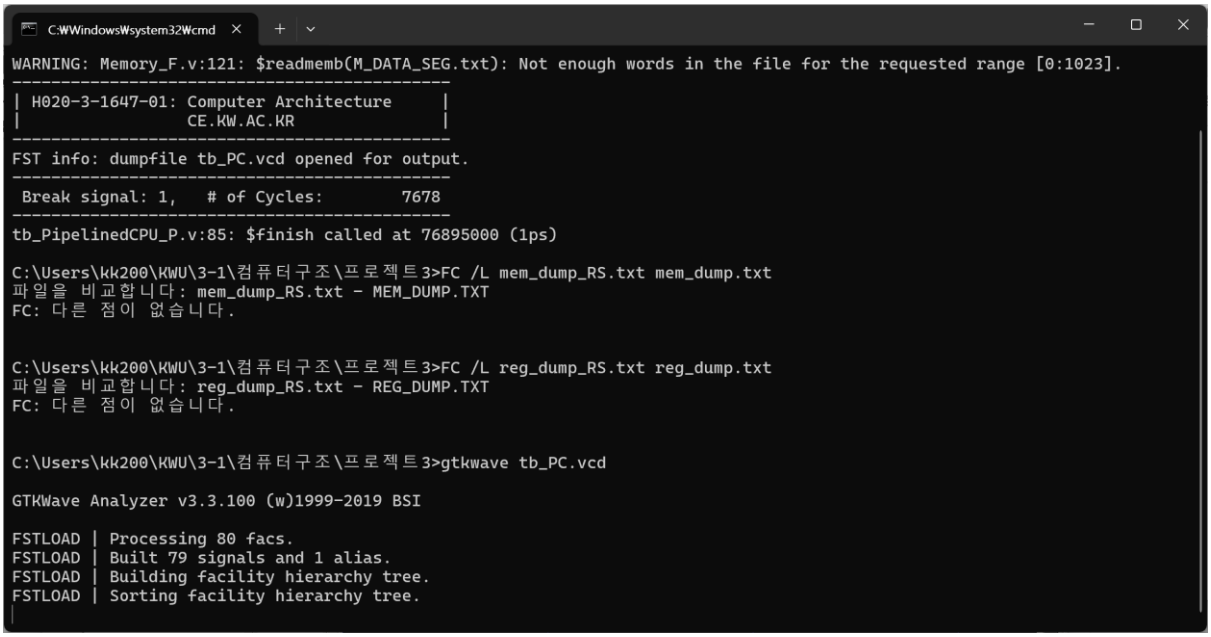
L3 add 와 L4 lw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L4 lw 와 L4 sw 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

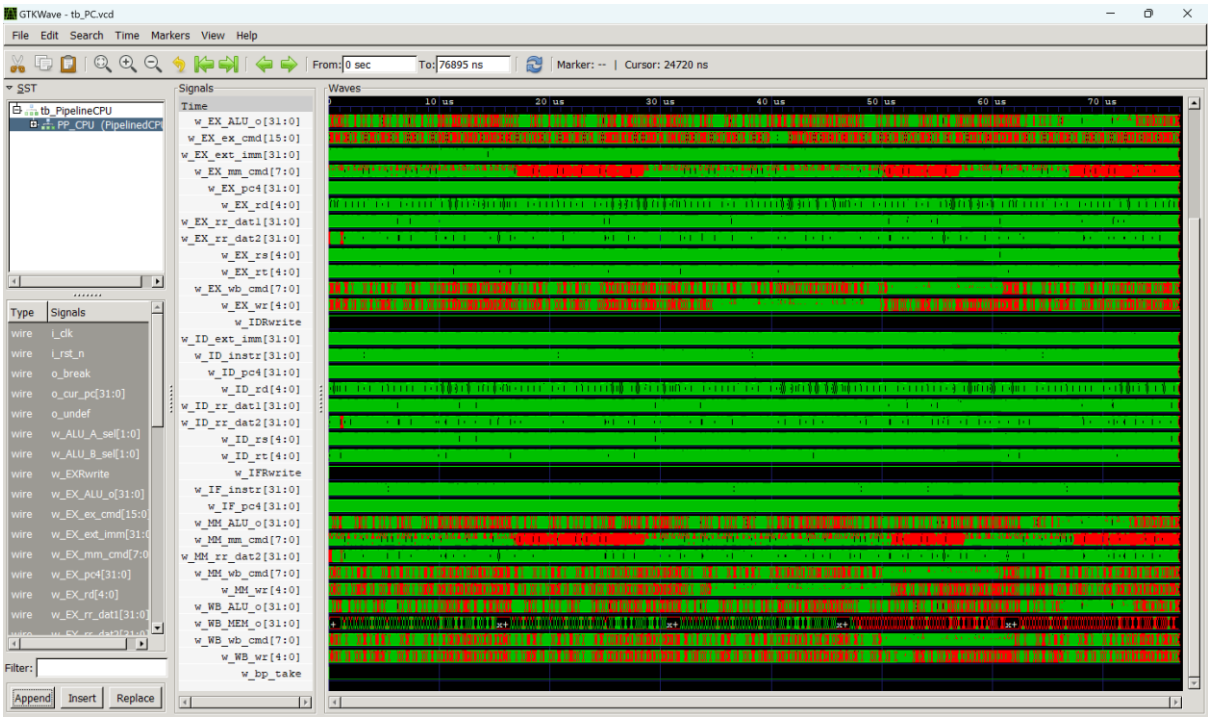
L4 addi 와 L4 slt 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

L5 add 와 L5 slti 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.

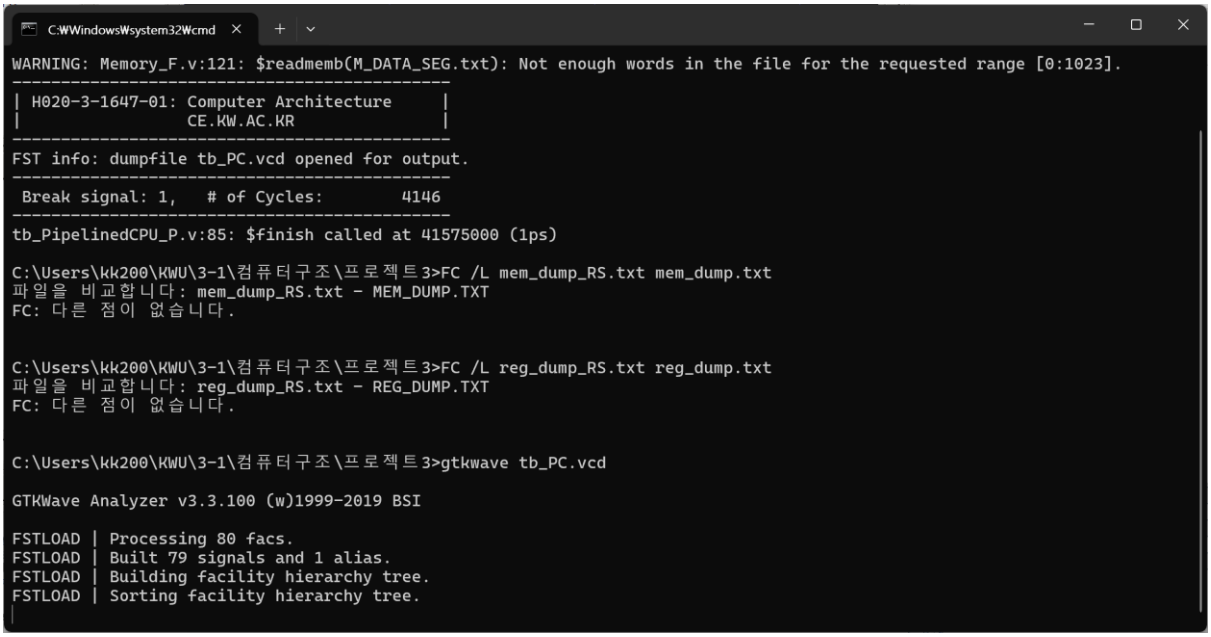
L4 add 와 L4 slt 사이의 data dependency 는 MM->ALU 로 해결할 수 있습니다.



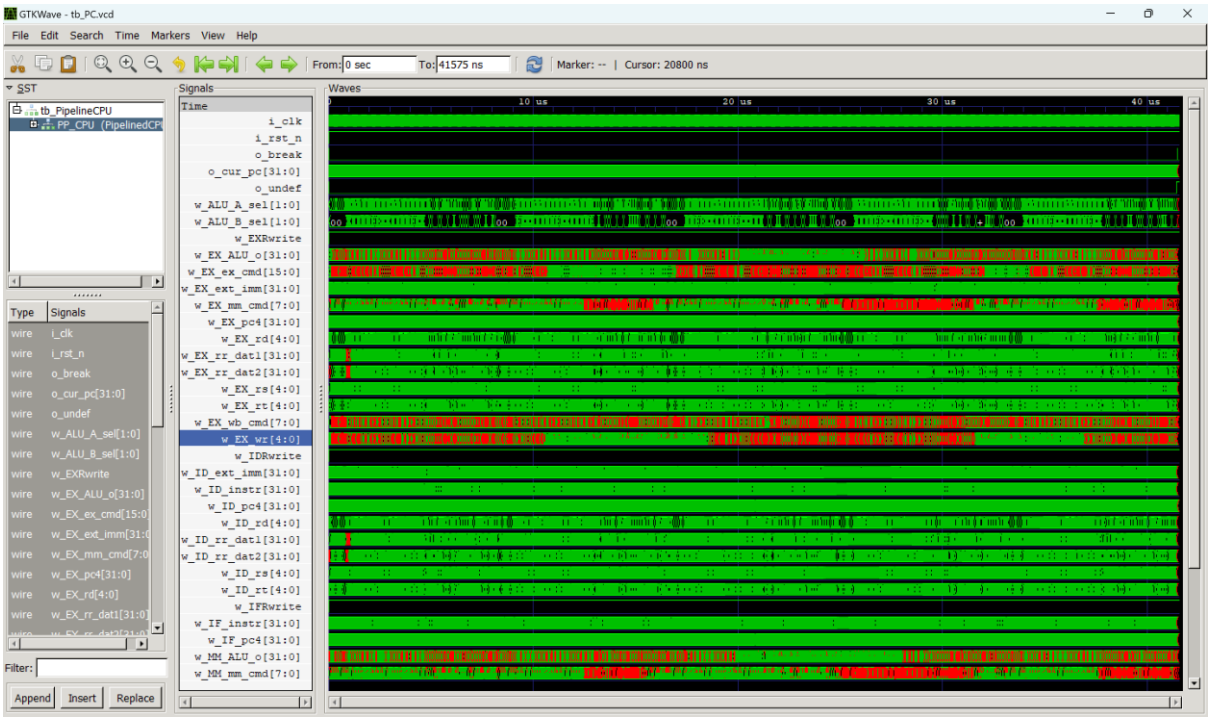
명령 수행에 걸린 총 cycle 수 : 7678 cycles



waveform 에서 한 사이클이 10ns 이므로 waveform 또한 7678 cycles 만에 프로그램이 종료됐음을 확인할 수 있습니다.



명령 수행에 걸린 총 cycle 수 : 4146 cycles



waveform 에서 한 사이클이 10ns 이므로 waveform 또한 4146 cycles 만에 프로그램이 종료됐음을 확인할 수 있습니다.

3. 고찰

forwarding 없는 data dependency 문제는 data dependency 가 존재하는 두 명령어 사이에 두 명령어와 data dependency 가 존재하지 않는 명령어가 3 개 이상 존재하게끔 nop 를 삽입해주면 해결되기에 문제가 되지 않습니다.

하지만 본 시뮬레이션은 정적인 forwarding 신호를 사용하기 때문에, forwarding 신호가 추가되게 되면 branch 명령어에 의한 jump 로 인해 각 Label 의 상단부에 존재하는 명령어들이 문제를 일으킵니다.

초반엔 branch 에 의한 문제를 고려하지 않고 과제를 진행하다 보니 많은 시행착오를 겪었습니다.

또한 각 Label 사이에 data dependency 가 존재하지 않도록 nop 를 삽입하는 중, Label 의 처음 라인과 마지막 라인 중 어느 쪽에 nop 를 삽입해야 하는가에 대한 문제를 마주하게 됐습니다.

여기서 제가 내린 결론은, branch 를 통해 jump 를 수행하면 Label 의 처음 부분부터 명령어가 실행될 것이고, 만일 nop 를 Label 의 처음 라인에 삽입한다면 jump 가 수행될 때마다 nop 가 반복적으로 실행될 것이므로, Label 의 마지막 부분에 nop 를 삽입하는 것이 cycle 면에서 봤을 때 더욱 효율적이다는 것입니다. 따라서 nop 를 Label 마지막 라인에 삽입했습니다.

Reference

컴퓨터구조실험 / 광운대학교 / 이혁준 교수님 / CA_Lab_10.pdf

컴퓨터구조실험 / 광운대학교 / 이혁준 교수님 / CA_Lab_11.pdf

컴퓨터구조실험 / 광운대학교 / 이혁준 교수님 / CA_Lab_12.pdf