컴퓨터구조 결 과 보 고 서 (Result report)

Major	Student ID	Grade	Name	Experiment date
융합전자공학부	2019047892	3	이한별	2023.06.16(금)
융합전자공학부	2021067501	3	김예림	
Title	Final Project			

1. 실험결과

1) 수정한 어셈블리 코드

Assembly Language Modify

```
0x00000093 addi x1, x0, 0
0x00000113 addi x2, x0, 0
0x00000193 addi x3, x0, 0
0x00000213 addi x4, x0, 0
0x00000293 addi x5, x0, 0
0x00000313 addi x6, x0, 0
0x00000393 addi x7, x0, 0
0x00000413 addi x8, x0, 0
0x00000493 addi x9, x0, 0
0x00000513 addi x10, x0, 0
0x00000593 addi x11, x0, 0
0x00000613 addi x12, x0, 0
0x00000693 addi x13, x0, 0
0x00000713 addi x14, x0, 0
0x00000793 addi x15, x0, 0
0x00000813 addi x16, x0, 0
0x00000893 addi x17, x0, 0
0x00000913 addi x18, x0, 0
0x00000993 addi x19, x0, 0
0x00000A13 addi x20, x0, 0
0x00000A93 addi x21, x0, 0
0x00000B13 addi x22, x0, 0
0x00000B93 addi x23, x0, 0
0x00000C13 addi x24, x0, 0
0x00000C93 addi x25, x0, 0
0x00000D13 addi x26, x0, 0
0x00000D93 addi x27, x0, 0
0x00000E13 addi x28, x0, 0
0x00000E93 addi x29, x0, 0
0x00000F13 addi x30, x0, 0
0x00000F93 addi x31, x0, 0
0x0004AE03 lw x28,0(x9)
0x00100313 addi x6, x0, 1
0x00100E93 addi x29, x0, 1
0x03CE8A63 case 1: beq x29, x28, exit
0x001E8E93 addi x29, x29, 1
0x006283B3 add x7, x5, x6
0x00700533 add x10, x0, x7
0x03CE8263 case 2: beq x29, x28, exit
0x001E8E93 addi x29, x29, 1
0x007302B3 add x5, x6 x7
0x00500533 add x10, x0, x5
0x01CE8A63 case 3: beq x29, x28, exit
0x001E8E93 addi x29, x29, 1
0x00538333 add x6, x7, x5
0x00600533 add x10, x0, x6
0xFD1FF06F j case1
0x00800F13 exit: addi x30, x0, 8
0x00AF2023 sw x10, 0(x30)
0x19000F93 addi x31, x0, 400
0x000F8F13 mv x30, x31
0x0040006F exit 3: j exit0
0xFFDFF06F exit 0: j exit3
```

2) 개선 방법

1_소프트웨어 개선

기본 피보나치 수열 어셈블리는 과도한 Branch 와 Load, Store Instruction 을 사용하였습니다. 이에 가장 많은 Cycle Time 을 소비하는 Load, Store 와 Branch 사용을 줄이는 쪽으로 개선을 시도했습니다. 이전 코드에서는 각 Iteration 을 Stack 에 쌓는 방법을 사용하여 수열 첫번째부터 계산을 시작하였습니다. 하지만, Stack 에 쌓인 수를 불러올 때에 해당 Stack 의 값을 0 이 될 때까지 반복 계산한 비효율적인 설계를 확인했습니다.

개선된 코드에서는 Load, Store 사용을 지양하고 최대한 Stall 이 발생하지 않도록 코드를 설계했습니다. 각 Stack 에 값을 할당하였던 것을 Temporary Register 사용했습니다. Register x5, x6, x7 을 Circular Pipe 구현에, x28, x29 를 각각 Limitation(=25), Iteration 용으로 사용해 Circular Pipe 개념에 입각하여 피보나치 수열을 계산하도록 설계했습니다. 그 결과, 아래 Figure02 와 같은 성능 향상을 이끌어냈습니다. 아래 Figure01 은 개선된 코드가 계산하는 방법을 표현했습니다.

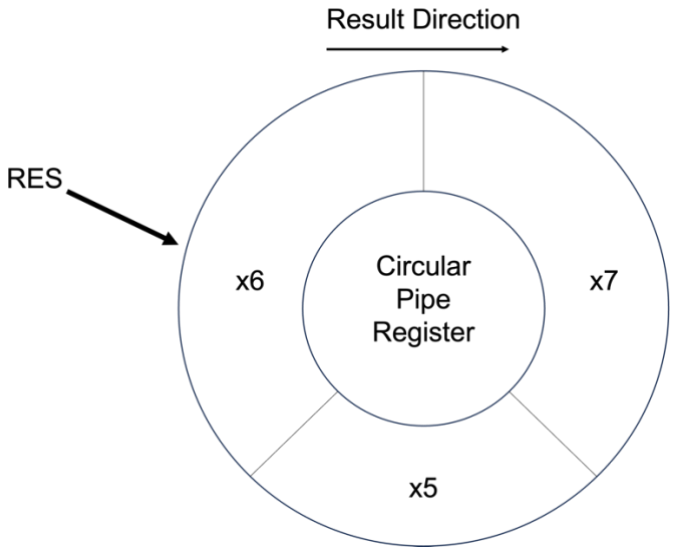

Figure01. Circular Pipe Register

x5, x6, x7 로 피보나치 수열을 계산하는데 사용하였고, x28 은 몇 번까지 수열을 반복해야 하는지, x29 는 몇 번 반복했는지를 저장했습니다. 먼저, 시작은 초기값을 지정해주기 위해 x6 에 1 을 저장했습니다. 그리고 Iteration 을 한 번 수행한 것으로 간주하여 x29 에 1 을 저장하였습니다. 그 후에 두 번째 수는 x7 에 x5(=0)와 x6(=1)을 더한 값을 저장하고 Iteration, x29 의 값을 1 증가시켰습니다. 다음은 Circular 규칙에 따라 x5 에 x6(=1)과 x7(=1)의 값을 더하여 세 번째 값(=2)을 저장한 후 Iteration 을 1 증가시켰습니다. 마지막 Case3 에서 x6 에 x7(=1)과 x5(=2)의 값을 더하여 네 번째 값(=3)을 구하고 Iteration 을 1 증가시켰습니다. 각 Case 마다 첫 번째 Instruction 으로 beq 을 추가하여 Iteration(=25)로 목표 수열 값까지 반복했으면 Exit 으로 Jump 하고 잘못 계산되지 않도록 하였습니다. 또, 각 값들을 최종적으로 저장해야 하기에 각 Case 별로 x10 에 값을 덮어쓰도록 "add x10, x0, (해당 결과를 저장한 레지스터)"로 구현했습니다. 이런 규칙성에 의거하여 Case3 까지 실행되면 무조건 Case1 으로 jump 하면 되기에 ' j Case1' Instruction 을 사용했습니다.

Circular Pipe 의 개념을 사용한 것은 레지스터에서만 계산할 때 저장 규칙을 단순화하기 위해서 입니다. 피보나치 수열은 현재 값을 구할 때 이전 값과 그 이전의 값을 더합니다. 이 점에서 세 개 공간을 할당할 수는 있지만, Circular 개념이 없다면 각 Iteration 별로 값들을 이동시키거나 Stack 을 사용해야하는 문제가 발생합니다. 반면 Circular 개념이 있다면 당연하게 값을 이동시키지 않고도 현재 값을 원활히 계산할 수 있습니다. 그렇기에 Circular pipe 개념을 도입하여 개선했습니다.

2_ 하드웨어 개선

기존의 계획은 Branch Condition 을 개선하기 위해 Branch Predictor 를 구현하고자 했습니다. 그러나 어셈블리 코드를 개선한 후, Branch taken 이 오직 1 회 발생하면서 Branch Predictor 로 개선해도 위 결과보다 극적인 개선 효과는 볼 수 없을 것이라 판단하여 SW 개선에 집중했습니다.

3) 개선된 결과