計算機組織

112學年度第2學期

Final Project

Pipelined CPU Design

老師： 朱守禮 老師

學生： 11127132羅海綺

11127137黃乙家

11127138林雨臻

一、背景

為了使學生充分了解，該如何使用Verilog HDL與Modelsim模擬器，以Midterm Project 所設計之 ALU Design及計算機組織課程講義：

[1] Chapter 4:The Processor

[2] Pipelined Datapath

為基礎，設計一個 Pipelined MIPS-Lite CPU。

二、方法

1.功能說明：本Project包含下列 16 道 MIPS 指令。

(a) Integer Arithmetic: add, sub, and, or, sll, slt, andi

(b) Integer Memory Access: lw, sw

(c) Integer Branch: beq, j, jr

(d) Integer Multiply/Divide: multu

(e) Other Instructions: mfhi, mflo, nop

2.設計要求：Datapath 與詳細架構圖(並以 PowerPoint 或 Word 設計繪製)。

(1) ALU:

(a) 設計重點說明：

* 需使用Midterm Project所設計之ALU完成add, sub, and, or, sll, slt, andi指令。

(2) Datapath:

(a) 設計重點說明：

* 所有指令之執行，須遵守 5-Stage Pipelined CPU 執行指令之行為。

(3) multu:

(a) 設計重點說明：

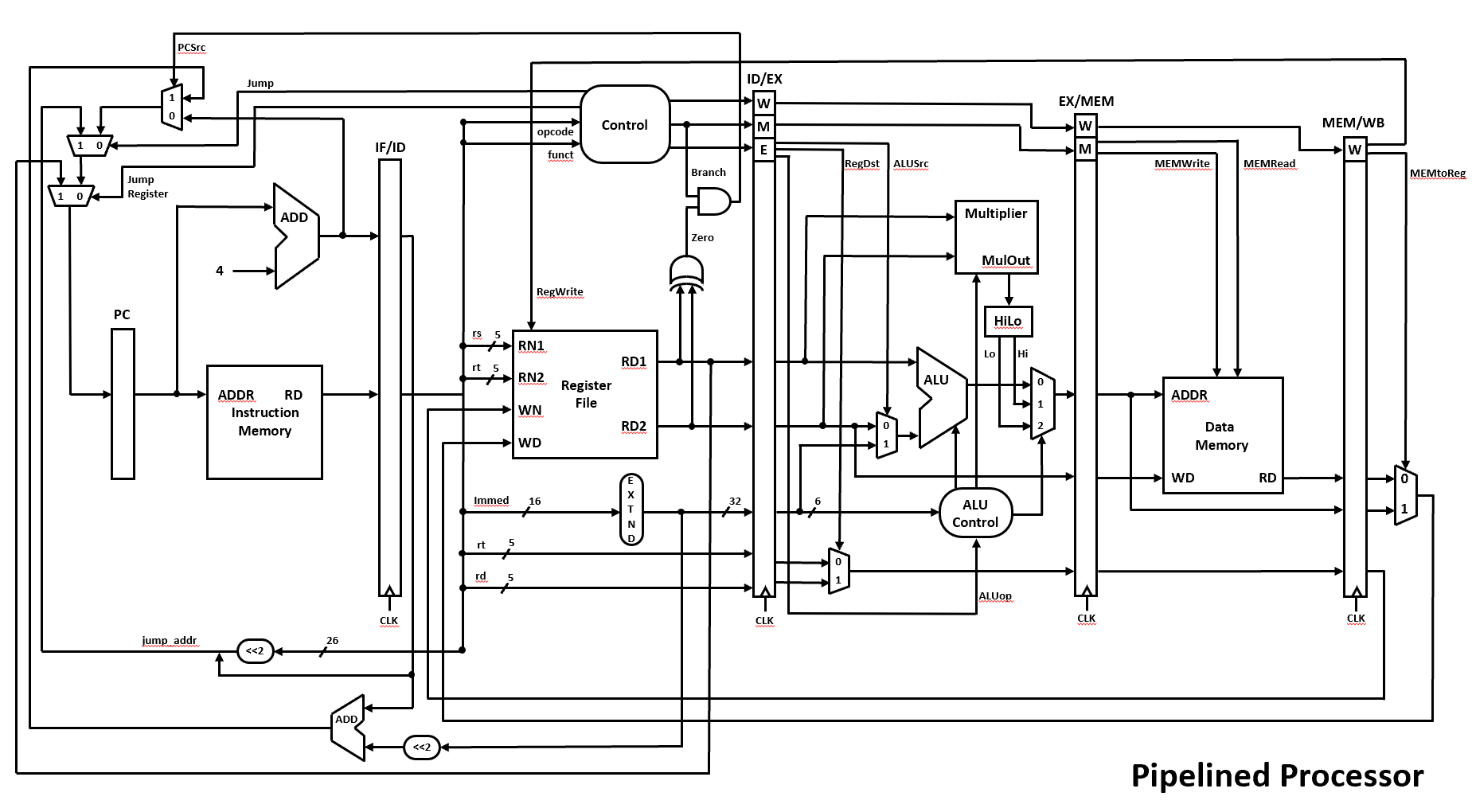
* 32-bits 無號數乘法指令。
* 需使用 Midterm Project 所設計之 Multiplier。

(5) Testbench:

(a) 設計重點說明：

* 須以讀檔的方式，讀入測試資料。
* Single Cycle 版本之 MIPS 處理器設計與Testbench Verilog 範例，請詳見 i-Learning 上之「Single Cycle CPU and Testbench」
* 本機測時將有不同的測試資料，以確認設計正確性與完整性。

(b) Datapath與詳細架構圖：請參考附錄(CO.pptx)。



程式架構：

mips\_pipeline:

IF - Instruction Fetch:

PC

PCADD (PC+4)

memory (InstrMem)

2-1 MUX\*3

ID - Instruction Decode:

ControlUnit

sign\_extend or unsign\_extend

RegFile

BEQ (從MEM移到ID)

2-1 MUX\*1

EX - Execute / Address Calculation:

ALU

ALUControl

Multiplier

HiLo

2-1 MUX\*2

3-1 MUX\*1

MEM - Memory Access (read write):

memory (DatMem)

WB - Write Back (result into register file):

2-1 MUX\*1

三、結果

(1) mips\_pipeline:

(a) 宣告線路:

 (i) instruction bus

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述(ii) break out important fields from instruction

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述(iii) datapath signals

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述(iv) control signals

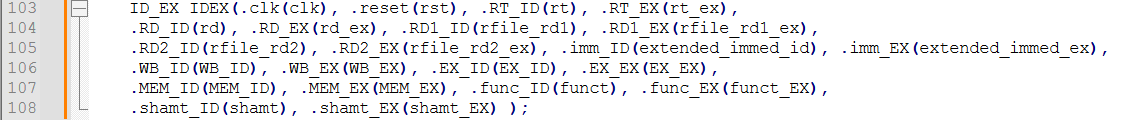
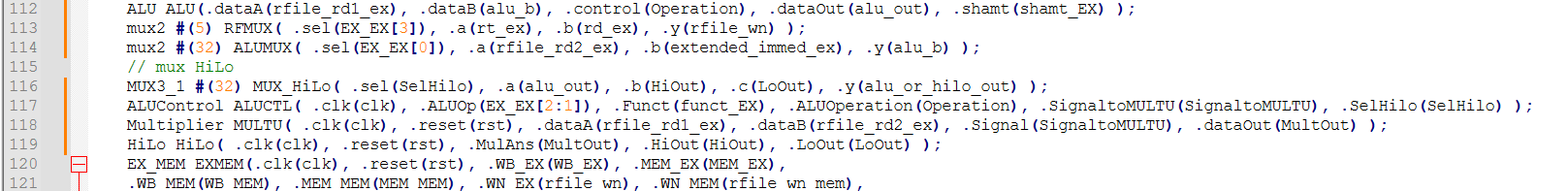
一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述(v) 分割線路

(b) Pipelining stages:

(i) IF - Instruction Fetch一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

(ii) ID - Instruction Decode一張含有 文字, 字型, 數字, 螢幕擷取畫面 的圖片

自動產生的描述

(iii) EX - Execute / Address Calculation

(iv) MEM - Memory Access (read write)

(v) WB - Write Back (result into register file)

(2) IF - Instruction Fetch:

(a) PC:

當時脈clk posedge時，若非reset就輸出算好的下道指令位置。

(b) memory:

MemRead或者mem\_array有變化時，判斷MemRead是否為1，若為1至指定記憶體位置讀取資料。

當時脈posedge且MemWrite為1時，將資料寫入指定記憶體位置。

(3) IF\_ID:

若非reset，傳遞instruction以及nextPC。

(4) ID - Instruction Decode:

(a) ControlUnit:

判斷OpCode，設定control signals。

(b) reg\_file:

當RN1或file\_array有變化時，判斷RN1是否為0，若為0直接賦值RD1=zero，否則RD1 = file\_array[RN1]。

RN2同理。

當時脈posedge時判斷RegWrite訊號是否為1且WN是否不等於0，若皆成立，寫入資料WN暫存器中。

(c) unsign\_extend:

無號數擴充直接補16’b0。

(d) sign\_extend:

有號數擴充將第15個bit補16次。

(e) BEQ:

我們將BEQ指令從MEM移到ID階，提早判斷並決定是否跳躍，減少nop數量從3個至1個。

實現方法：先判斷opcode是否為BEQ，再比較兩個基底暫存器的內容是否相等，若成立則將zero設為0。

(5) ID\_EX:

若非reset，傳遞訊號至下階段。

(6) EX - Execute / Address Calculation:

(a) ALUControl:

用funct決定ALU訊號。

(b) ALU:

ALU 32位元做AND、OR、ADD、SUB、SLT、SLL、將乘法器64位元的運算結果分割成Hi、Lo、移位器左移運算。

(c) Multiplier:

執行期中project的乘法運算。

(d) HiLo:

若非reset，將HiLo設為乘法運算的結果。

(7) EX\_MEM:

若非reset，傳遞訊號至下階段。

(8) WB - Write Back (result into register file):

用一個2-1MUX判斷是否要寫回reg\_file。

四、討論

(1) Modelsim 驗證結果：

1. // lw $s1(17,2), $t7(15,21), 0

00

00

2F

8E

// nop

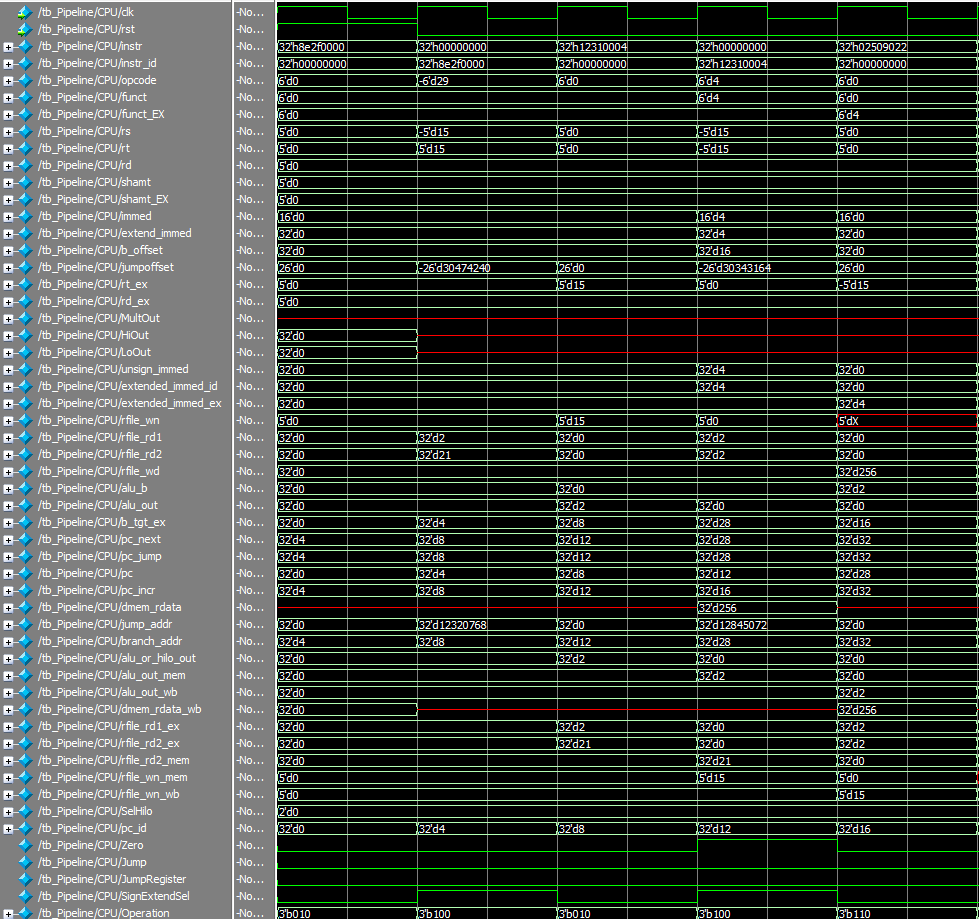
00

00

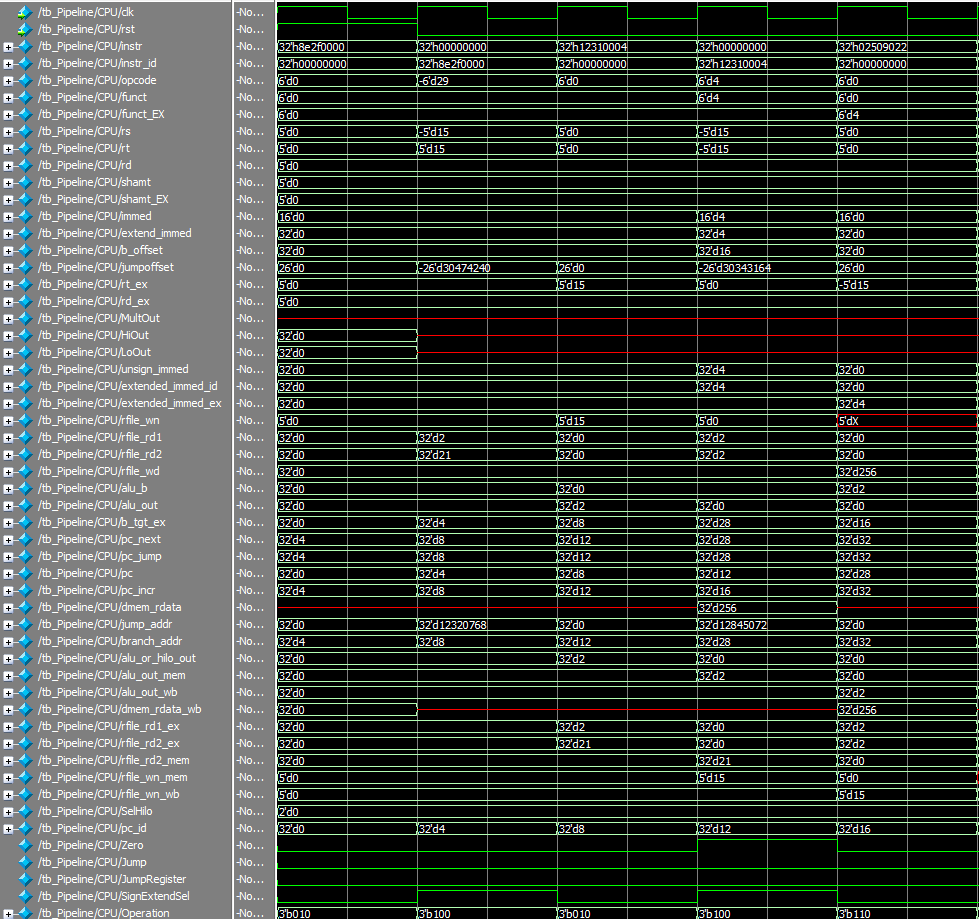
00

00

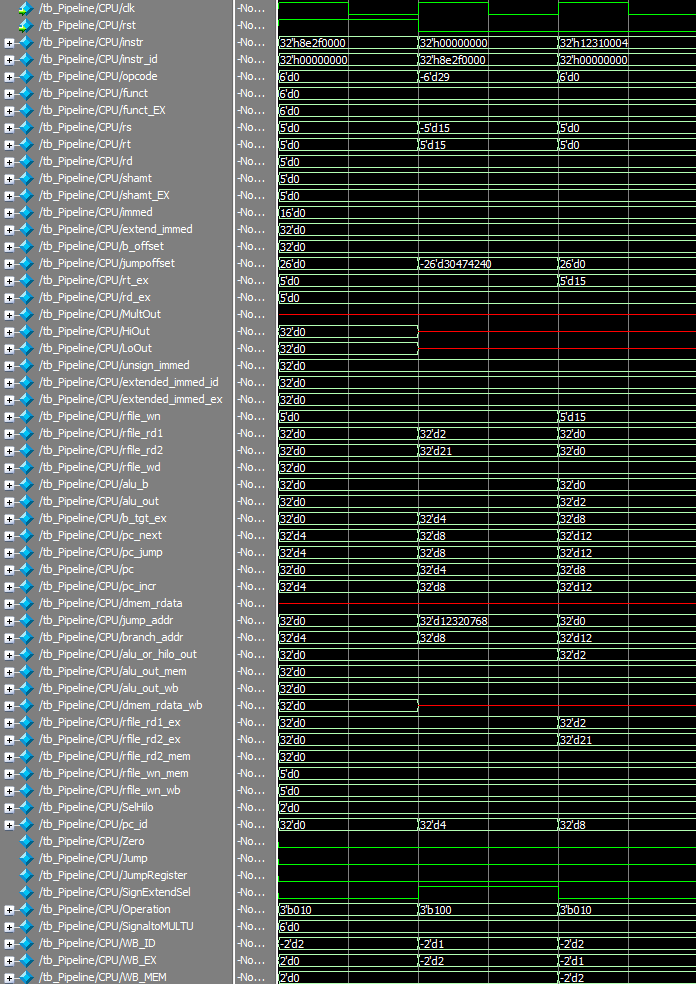
第一個cycle先fetch指令，因為電腦以little Endian的方式讀取，所以顯示8E2F0000



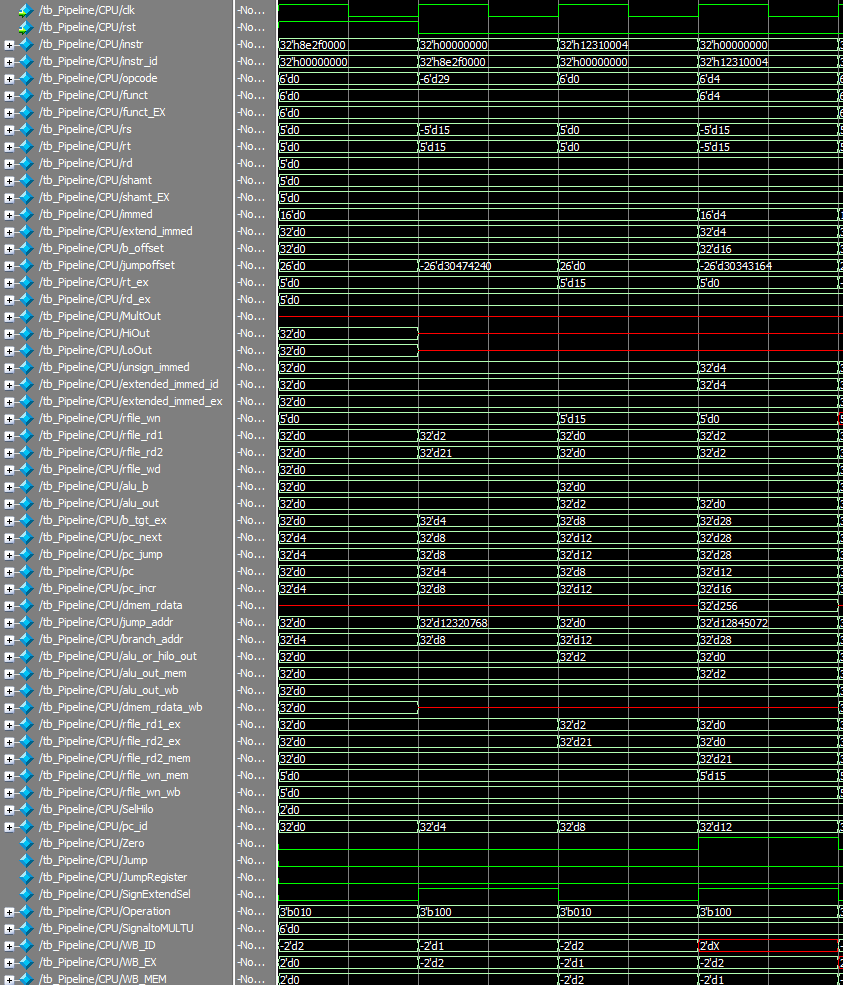
第二個 cycle 將lw指令解碼，並fetch nop指令，因為I-type只有兩個暫存器，而lw只用到 rd1，因此我們這時只需要看rs的位置與rd1的值。

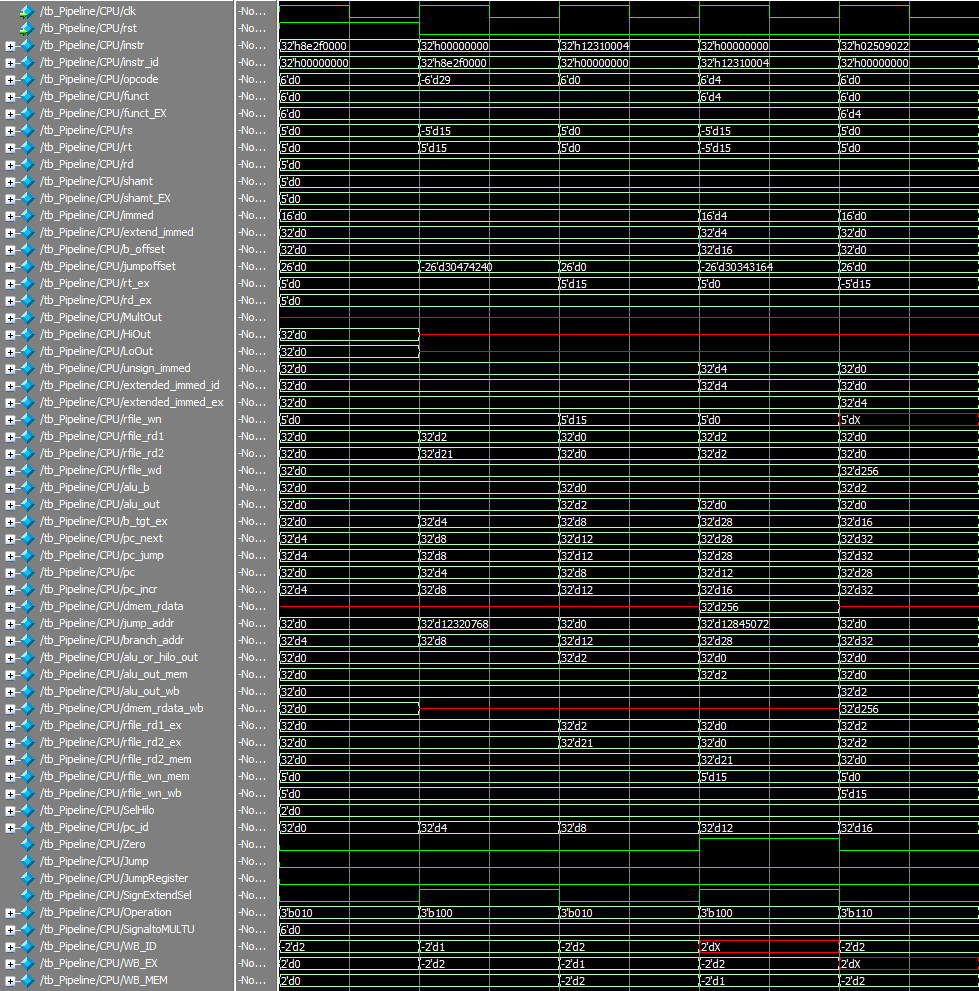


第三個cycle，lw進行運算，我們給的立即值為0，因此ALU 進行運算後的位置為2，同時nop進行解碼。



第四個cycle，lw進行memeory access，我們從ADDR讀到暫存器的位置=2，得到的記憶體的資料=256。



第五個cycle，lw將計算後暫存器write back to register file的wn(rt)。

2.// beq $s1, $s1, 4

04

00

31

12

// nop

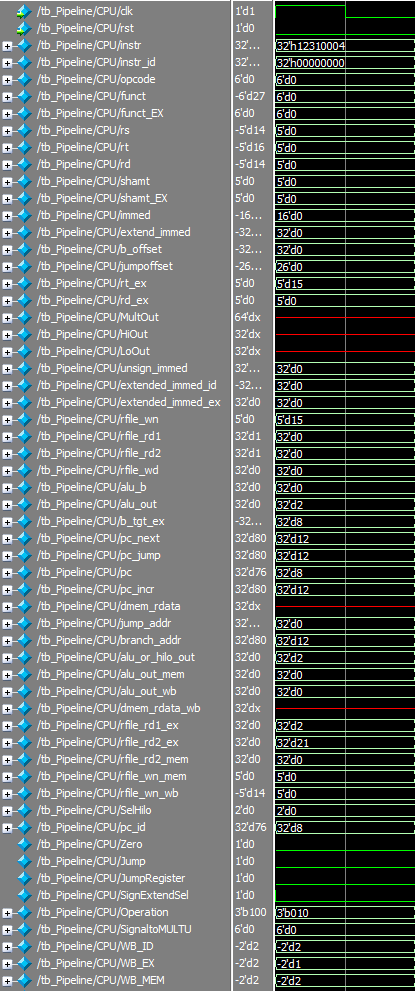
00

00

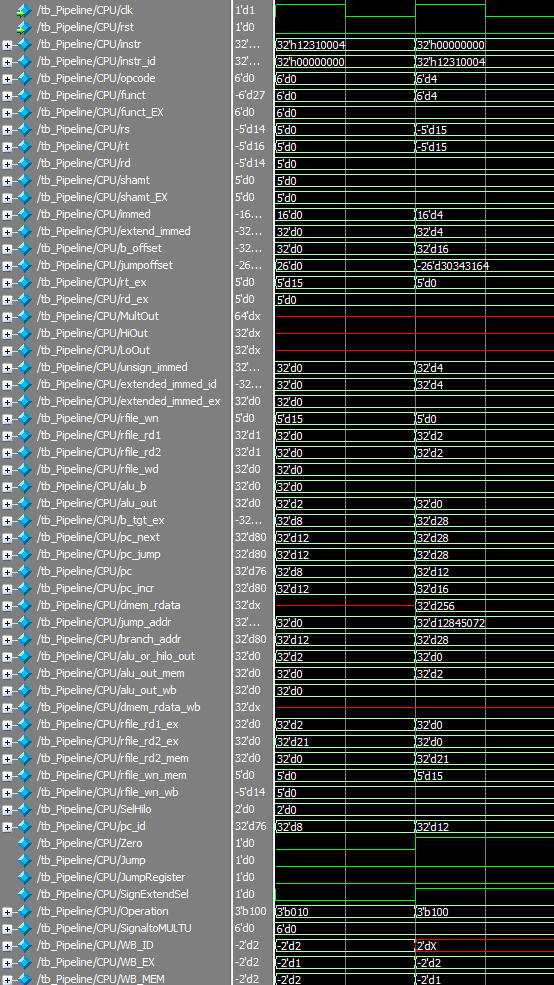
00

00

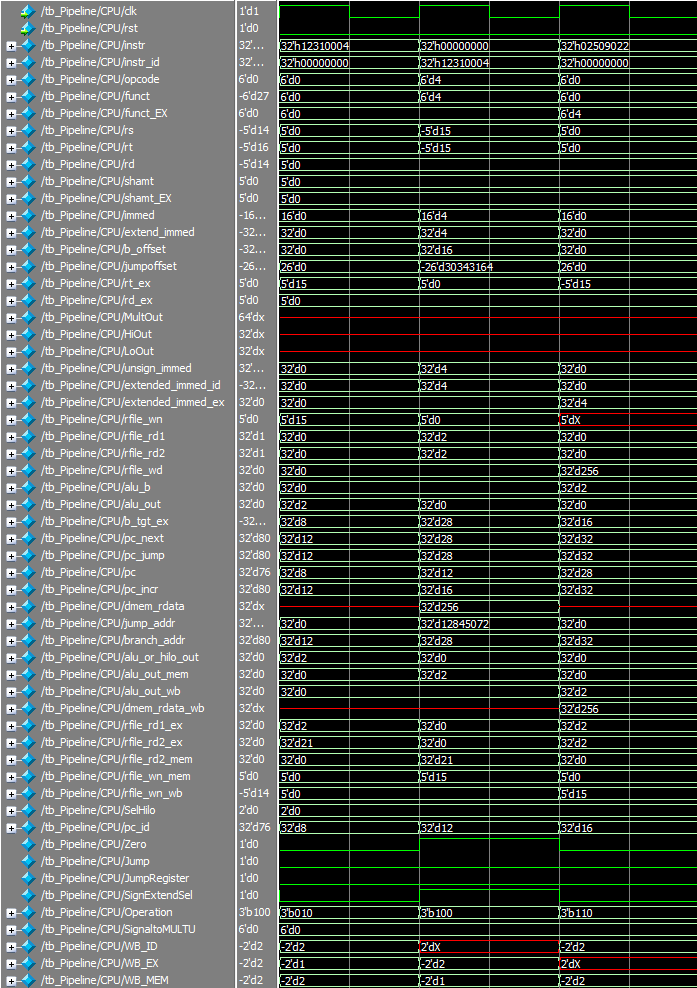
第一個cycle先fetch beq指令。



第二個cycle將beq指令解碼，fetch nop指令。



第三個cycle判斷rd1、rd2值是否相等，由圖可知相等，偏移量為4， offset為16，將 offset與 PC+4 相加，得出28，回傳給pc。



3.

// sub $s2(18), $s0(16,1), $s2(18,3)

22

90

50

02

// nop

00

00

00

00

// nop

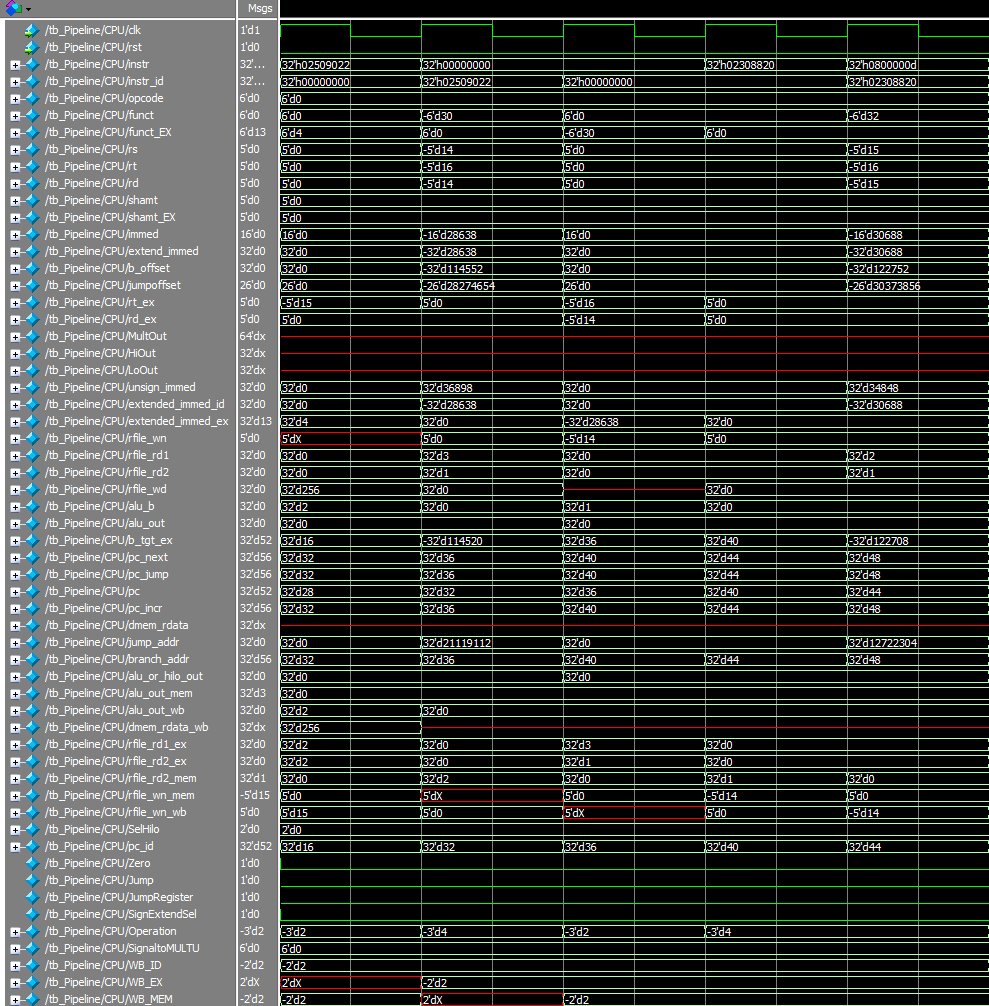
00

00

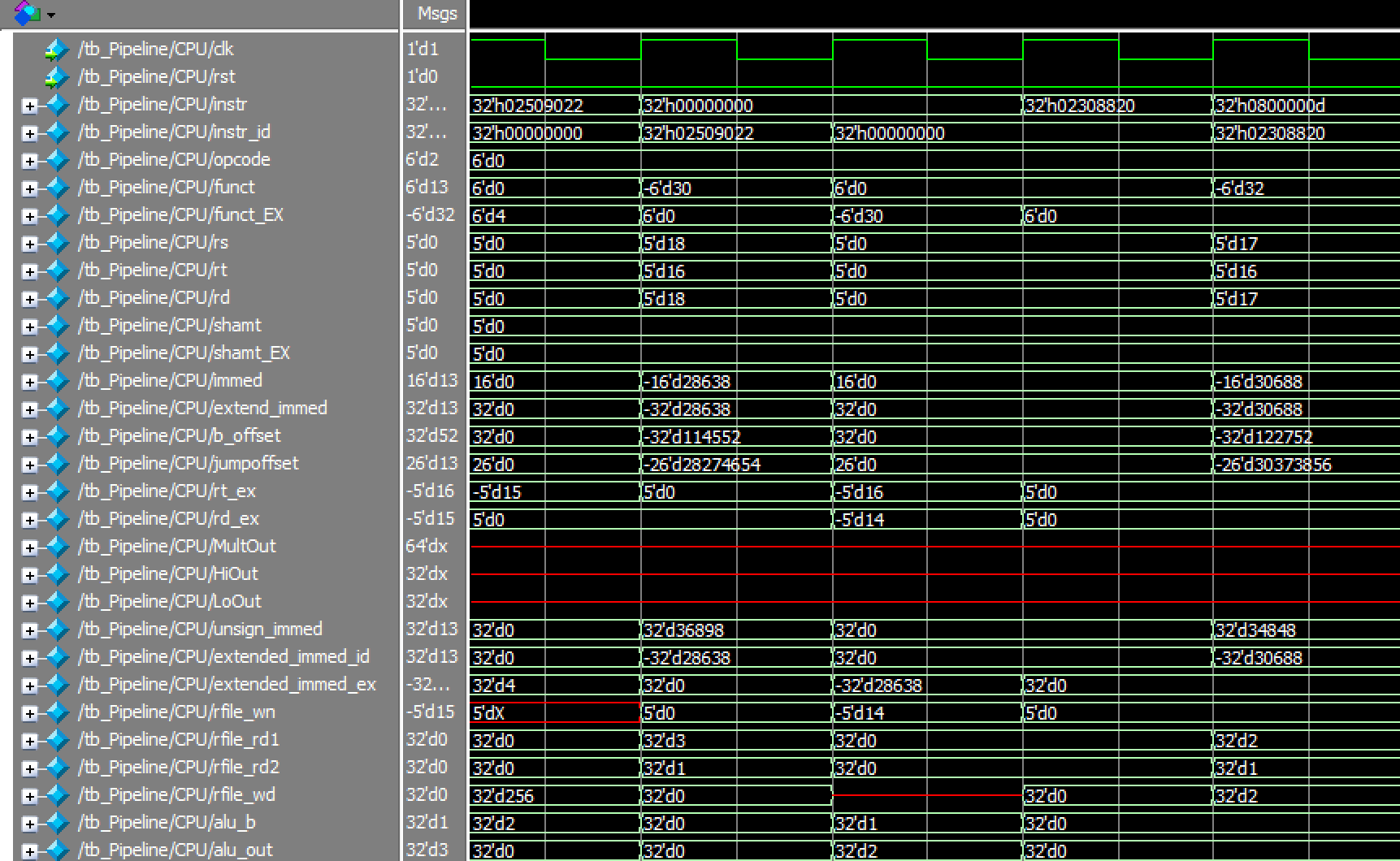
00

00

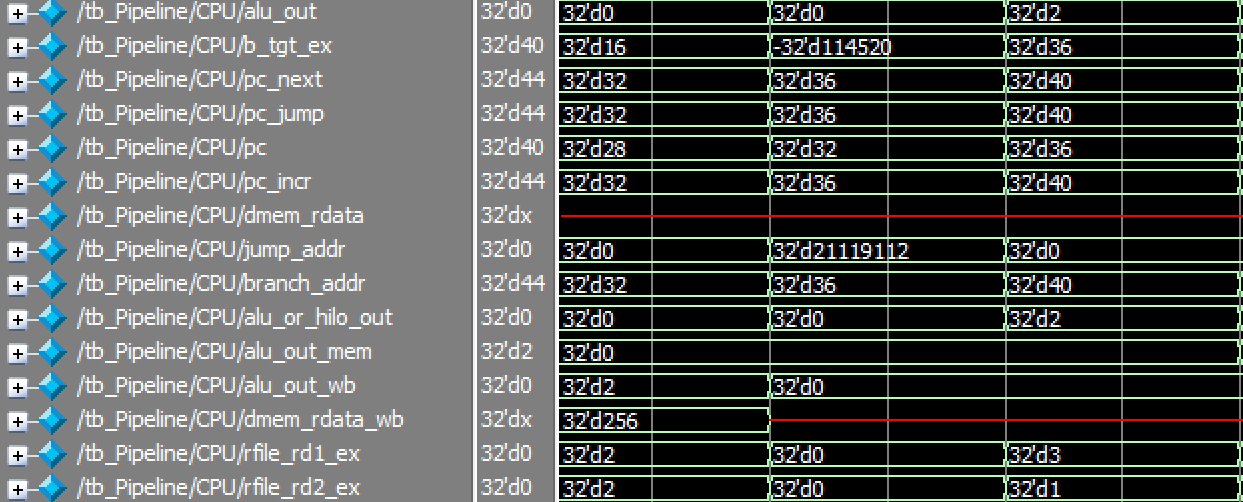
第一個cycle先fetch sub指令。



第二個cycle會將指令解碼，找到rs、rt的位置，rd1、rd2為暫存器的值。並fetch nop指令。

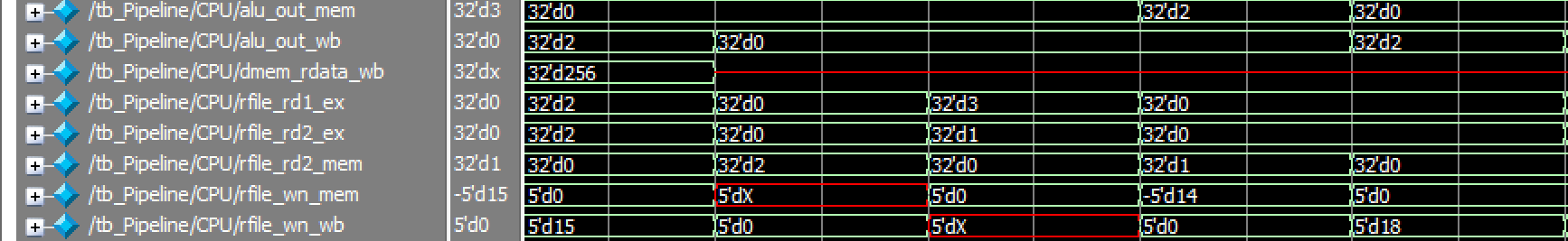


第三個cycle會將rd1、rd2放入ALU進行運算，得出2。



第四個 cycle，由於R-type不需要經過data memory，但還是需要傳遞。

第五個 cycle，要將ALU計算完的值write back to rd，這時rd暫存器的位置為18，值為2。



4.

// j 13

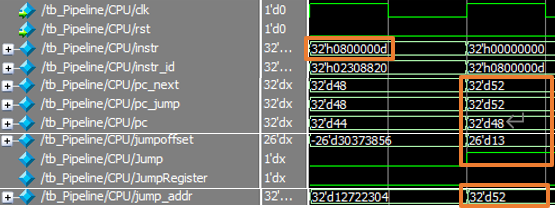
0D

00

00

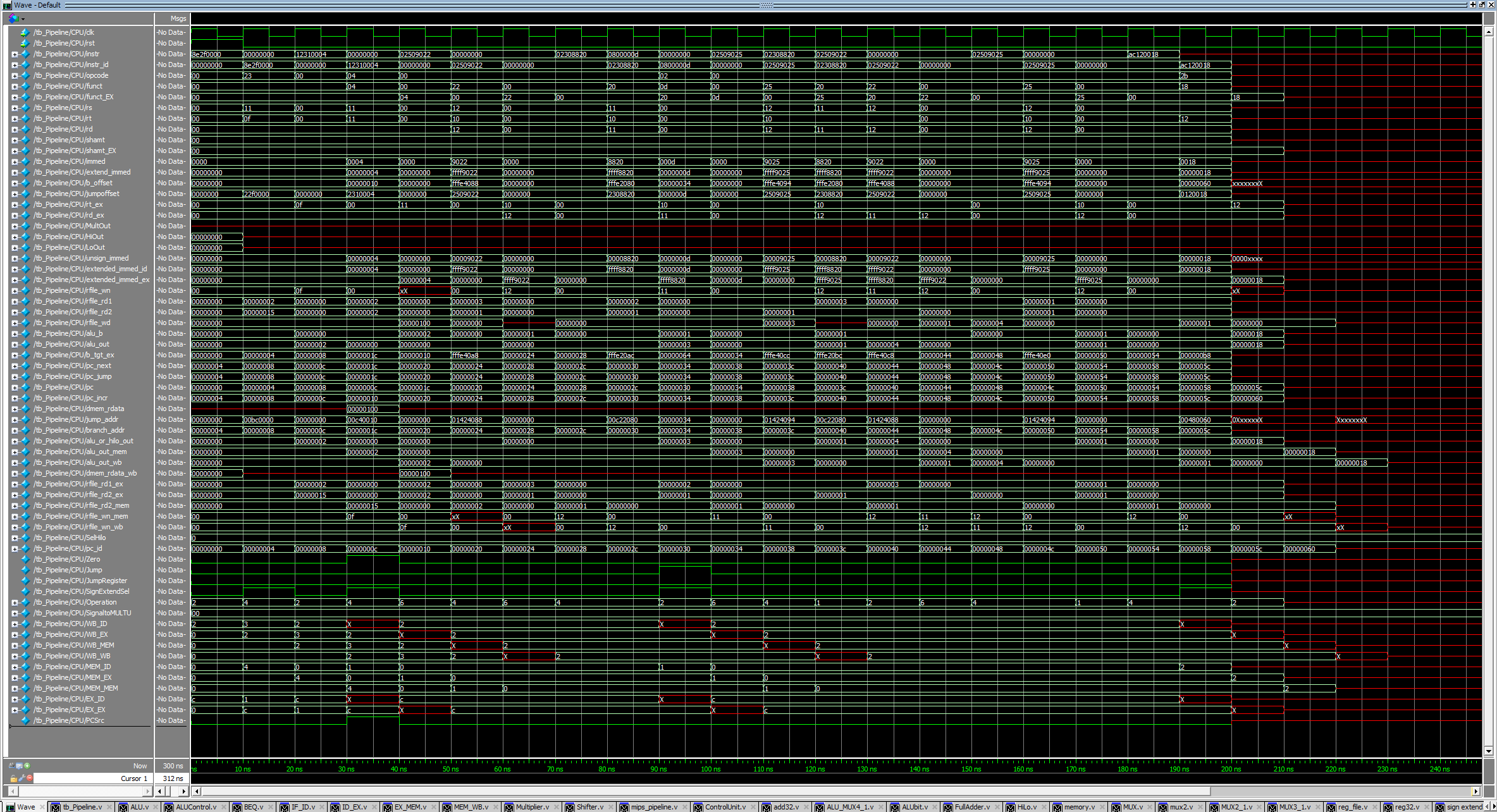
08

第一個cycle fetch jump指令，第二個cycle將jumpoffset與pc+4[31:28]concat，得出jump\_addr=52，同時control Unit會傳出jump指令給jumpMUX去選擇，最後將pc\_next設為52。



1. 結論

完整波型圖：



各組員分工方式與負責項目：

1. 分工方式：

組員討論寫法後，一人寫一支程式，其他人透過加註解及撰寫書面報告

的方式理解程式，為求所有人都理解程式內容。

1. 負責項目：

羅海綺：書面報告(驗證程式)、畫圖、部分module

黃乙家：mips\_pipeline、串接所有程式、debug

林雨臻：書面報告(模組說明)、部分module

心得感想：

這次的project要實作MIPS 5-Stage Pipeline Processor ，此次的架構是基於期中所寫的模組做擴充而完成的，本來我們認為應該只需要多寫一些串接各階段訊號的module 再加一個統整的module 就可以完成了，但實際寫後卻發現其實並沒有想像中的那麼簡單，因為課本中的圖並沒有完整畫出所有指令的訊號傳遞方法，尤其是我們這屆的特殊指令jr跟andi 因為之前對他們的使用並不了解，因此我們需要上網查其他架構圖，再逐一推算出指令的傳遞路線跟每個signal的值 。

在驗證波形圖時，展開後那一長串的訊號常常令我們看的眼花撩亂，且也常會發現有線路少傳遞或者指令進行了不對的運算，而此時就需要回去將相關連的module 都再看過一次尋找錯誤的來源，或者再加入少傳遞的訊號，並將那部分指令的內容補齊，且因為電路是同步運行的，因此雖然不是每到指令都會用到所有signal，但他們卻會在電路中被同步改變，在debug時要回來對訊號時，就要顧及出現錯誤的地方可能是之前的哪道指令造成的。

這次的project遇到hazard問題時，雖然只需要使用軟體解法就好，但因為我們一開始以為要需要硬體解法，因此多繞了不少彎路，此外我們還發現儘管已經加入nop進instr了，但執行乘法器時乘數跟被乘數的值卻會被nop蓋掉，發生data hazard ，這才發現期中時所寫的乘法器除了最初發現的不會被重置的問題之外，還需要將 counter寫入乘法器中，使得32次運算都直接在乘法器中進行就好，否則在訊號傳遞的過程中因為接收到新的值(nop)RN1跟RN2都會被改成0。

此次的project使我們學習到了很多，雖然並沒有實作硬體解hazard的方法，但卻還是具有一定的難度，但不得不說，成功完成後的成就感會讓我們覺得自己的努力並沒有白費。