Branch/Jump Delay Slots and ICM and DCM in a medium delay branch 5-stage pipeline:

Page 3 (next page) contains (Just FYI) solution to the Fall 2013 Midterm question on implementing one branch delay slot on an early branch design considering ICM (Instruction Cache Miss).

Page 4 contains (Just FYI) Late Branch Design

Page 5 contains a Medium delay Branch Design with an incomplete support for JUMP, ICM, and DCM. Here we want to execute both the branch and the jump from the EX stage (which is the medium delay design). This diagram is dense. So do not work on this. Just read how a late branch was converted to an early branch. We copied the page 5 design and copied on to the page 6 and removed several items on page 6 to make space for you to work on. Page 7 and page 8 are identical to page 6 and contain the same incomplete design.

1.1 On Page 6, complete the design to support JUMP, ICM and DCM assuming ZERO delay slots.

1.2 On Page 7, show only changes needed to convert your 0-delay slot design of page 6 to a 1-delay slot design.

1.3 On Page 8, show only changes needed to convert your 0-delay slot design of page 6 to a 2-delay slot design.

1.4 Conflict or cooperation between a successful branch and HDU/HDU_Br and how presence or absence of one or more delay slots affects this? (Use the words: defying / acts like a guardian angel)

Early branch design: Here HDU-Br acts like a guardian angel and protects a branch from acting based on incorrect data. Presence or absence of the one delay slot does not change the above.

Late branch design: If there is no delay slot, the HDU initiated stall works against the branch in Mem who is senior to both its Inst and the lw in EX. This is defying!

But if 2 delay slots are declared, we do need to preserve and safeguard the interest of the junior in the EX stage here, HDU will never initiate a stall when branch is successful.

So the question of defying or protecting does not arise.

1.5 It is easier to fill ____ jump ____ (branch/jump) delay slots because ____ jump is unconditional.

So, the compiler can bring the first instruction in the target area and put it in the delay slot. He does not have to check if it is safe because it is not based on prediction.

1.6 If compiler designer put a NOP (no operation instr.) in every delay slot, then he is causing branch penalty whether the branch is successful or not. This is worse than the hardware flushing because the hardware flushing occurs only if the branch is successful.

Program runs correctly but less efficiently compared to hardware flushing, etc.

1.7 LW delay slot eliminates HDU in our 5-stage pipeline T F. ___ (LW delay slot)

Branch delay slot) (simplifies eliminates) HDU_Br in our early Branch design.
When ICM = 1 we send a bubble into the next stage ID except when there is a Jump or successful Beq. This exception is taken care of through stalling the IF/ID which overrides IF_Flush.

ICM causes PC to be stalled.

ICM = 1 together with a jump or a successful Branch or a Jump in ID stage should cause to stall the Jump or Beq in ID until the delay slot instruction is available (until ICM = 0).

Jump should override any spurious stalls initiated by HDU.

Jump address injection through an intercept-and-inject mux.

When a Jump or a successful Branch is stalled in the ID stage by the ICM, we inject a bubble into the EX stage.
Medium Delay Branch -- incomplete Jump Support

Do not show your work on this page -- Show it in the next 3 pages
Medium Delay Branch -- Complete the incomplete Jump Support, ICM, DCM, and the rest of the design assuming zero delay slots.

Hazard detection unit

PC

Instruction memory

Control

MemRead

Hazard detection unit

Branch

JBR1

DCM

ICM

IF.Flush

ME

WB

Rel内存

Data memory

VR

BR1

JUMP

DCM stalls the entire pipe. ICM stall the IF stage (i.e. the PC) and injects a bubble into ID stage. A successful branch or a jump should be able to override a simultaneous stall by ICM. There is no conflict between Beq or Jump and HDU as there is no LW in EX stage if a Beq or Jump is in EX stage. Should DCM stall prevail over Jump or a successful branch? It is a little inferior if the DCM blindly stalls the entire pipe. Here by letting the Branch or Jump finish, we gain some advantage because of overlapped work done on IF end of the pipe.

For grading purpose, it is OK if students stalled the entire pipe based on DCM.

Graders: I expect our undergraduates to use readable logic (the way I draw sometimes negative logic gates) but many graduate students may be struggling to figure out the gates they need to use to arrive at the logic they are trying to implement. Be watchful. Thanks!

Medium Delay Branch -- Complete the incomplete Jump Support, ICM, DCM, and the rest of the design assuming zero delay slots.

ICM should stall the PC (there by stall the IF stage) and send bubbles into the ID stage only if there is no JUMP or successful branch in the EX stage. This because in the presence of a jump or a successful branch, we are no more interested in the instruction that incurred the cache miss.

No need to prevent bubble injection when DCM is true!

Some students think that JBR1 should intervene possible stalling by HDU. The fact is that HDU will not initiate a stall if JBR1 is true, because now EX is not occupied by a LW!
10/31/2015

Improvements to my initial solution of 10/30/2015 for the 1-delay slot and the 2-delay-slot designs based on student input. Thanks to the following two students for their very thoughtful design improvements.
1. Rishabh Ghai <rghai@usc.edu>
2. Gowtham Chandrasekaran <gowthamc@usc.edu>

Deficiency #1 in my initial designs for 1-delay slot and 2-delay slot designs

We need to consider the case when a JUMP or BRANCH is in the ID stage when (in the same clock) an ICM occurs in the IF stage. In my current designs, ICM injects a bubble into the ID stage and the Jump or Branch in ID stage progress to EX stage. This effectively means that we (the hardware engineers) rewrote the code generated by the compiler by inserting a bubble (a NOP) between the Jump/Branch and their delay slot instruction(s). This is wrong because then we are not satisfying the intent of the programmer or the compiler. So the ICM in IF should check to see if the ID stage is occupied by a Jump or a Branch (no question of successful or unsuccessful arises when the branch is still in the ID stage) and hold him by his collar (or shirt) at that point itself (in that clock itself).

Deficiency #2 in my initial design for the 2-delay slot design

It is not enough for the ICM in IF to hold the senior successful Branch (underline the word successful) in EX stage by his collar until he (the ICM) is done. We need to stall the entire pipe because the successful branch in EX may be currently receiving help from a senior in the MEM stage or in the WB stage or from both seniors. If we hold just the branch in EX stage only and let the helping senior(s) leave, then the branch which was saying that he was successful, may say in the subsequent clocks that he is not successful because his help is gone. To avoid this problem, we need to stall the entire pipe. This applies to successful branches only. It does not apply to an unsuccessful branch because then the delay slot instructions and the rest of the fall-through instructions will execute eventually when ICM is done and this does not require any holding of the unsuccessful branch or the entire pipe. What about the Jump? Since the Jump can not be dependent on his seniors, if JUMP is in the EX, the ICM needs to just hold him but does not need to hold the entire pipe.
Implementation of fixes

Fix for Deficiency #1

It is rather unnecessary to override this stalling line by JBR1 because if a Jump or Branch is in the EX stage, the ID stage cannot possibly have a Jump or Branch in it. Recall that delay slot should not be filled with another control instruction!

ID_J_or_B in ID and JBR1 in EX are mutually exclusive.

Injects bubble into the EX stage when IF/ID is stalled.

Fix for Deficiency #2
Medium Delay Branch -- show only changes needed to convert your 0-delay slot design to a 1-delay slot design.
1-Delay Slot: Do not flush ID instr. but flush the IF instr. if BR1 or Jump = 1

Deficiency #1 is fixed here.

Medium Delay Branch -- show only changes needed to convert your
0-delay slot design to a 1-delay slot design

We just need to remove the ID.Flush to avoid flushing the
ID instr. Rest all same.

From the ID stage
Medium Delay Branch — show only changes needed to convert your 0-delay slot design to a 2-delay slot design.

Hazard detection unit

Implementation: Produce HOLD_JBR as (JBR1 & ICM). HOLD_JBR should disable the IF/ID and ID/EX stage registers so that the ID and EX instructions are held.

Unlike in the 0-delay slot design, here under DCM, we need to stall the entire pipe. So let us redraw the disable logic for the PC, the IF/ID, and the ID/EX.
2-Delay Slot — Do not flush the two juniors even if BR1 or Jump = 1
So remove the ID.Flush and IF.Flush

Medium Delay Branch — show only changes needed to convert your
0-delay slot design to a 2-delay slot design

Hazard detection unit

Implementation: Produce HOLD_JBR as (JBR1 & ICM). HOLD_JBR should disable the IF/ID and ID/EX stage registers so that the ID and EX instructions are held.

Unlike in the 0-delay slot design, here under DCM, we need to stall the entire pipe. So let us redraw the disable logic for the PC, the IF/ID, and the ID/EX.

From the ID stage
1.8 Briefly narrate the common aspects among the above three designs and also the differentiating aspects. Use words like: "inject bubbles", "Wrist-Band", "Hold it until ICM/DCM is over", "Beq or JMP will walk away like a bubble", "wasteful but harmless", "stall overrides bubble-injection", etc.

In all 6 DCM stalls EX/MEM and MEM/WB stage registers; ICM attempts to inject bubbles into ID and also attempts to stall the PC; HDU attempts to stall the PC and IF/ID; HDU attempts to inject a bubble into EX stage.

Differentiating: 1. O-delay slot: both IF-Flush and ID-Flush were present; 1-delay slot: ID-Flush was removed; 2-delay slot: IF-Flush was also removed.
2. In the 2-delay slot case, ICM holds the next two instructions on the delay slots.

Blank area: You can use it as rough or as additional space to complete Q#1.4 :)

Graders, this question is kind of open-ended and does not have a deterministic answer. So, while grading, try to be appreciative rather than punitive. Do not expect students to find all the similarities and differences that I thought are significant.

Thanks
Gandhi

1.9 In reality the instruction fetch stage and the rest of the stages (i.e. Instruction decode onwards) are separated by an Instruction Prefetch Queue (IPFQ) which is a FIFO.

FIFO ___________ helps (does not help) to even out the sporadic delays in production and consumption. Since on and off the Instruction cache incurs cache misses, it ___________ makes (does not make) sense to fetch 2 or 4 of instructions from the I.Cache and deposit into IPFQ.

Who writes into it (into the IPFQ)? I.Cache

Who reads from it (from the IPFQ)? ID stage

Who flushes it and why? Jump or a successful branch. Jump includes JAL and JR

Who needs to wait if it is full? I.Cache

Who needs to wait if it is empty? ID stage

Mr. TROJAN (Bruin/Trojan) suggested an 16-deep IPFQ.
Mr. BRUIN (Bruin/Trojan) suggested an 16K-deep IPFQ.

Explain your answer to the above questions regarding depth.

Branches occur so frequently and even to maybe big. If you are fetching a block of words at a time from I-cache, you can theoretically argue that you may have a series of hits but then you are forgetting that some branch or jump would flush if any ways.

1.10 JAL delay slot is ___________ similar to ___________ different from the jump delay slot.

JR $31 delay slot is ___________ similar to ___________ different from the jump delay slot.

CISC processors ___________ do not (also / /do not) use ___________ any (Branch/Jump/LW any) delay slots.
2. Given on the side is a short extract from your FIFO lab Verilog code

\[\text{fifo_reg_array_sc_n_plus_one_bit_pointers.v} \]. Let us assume that our Verilog compiler does not support the \text{always (*)} construct. Rewrite it below to suit.

\begin{verbatim}
always (@(*)
begin
  empty  = 1'b0;
  full   = 1'b0;
  if (depth == N_Plus_1_zeros)
    empty  = 1'b1;
  if (depth == A_1_and_N_zeros)
    full   = 1'b1;
end
\end{verbatim}

2.2 In your lab 7 Part 1 (3-element adder where an instruction turns itself into a NOP if there is an overflow), we have two comparators for Z source. You would carry the signal \text{ID_ZMEX1 / ID_ZMEX2} to the EX2 stage to produce \text{Z_FORW2} to control the select line on \text{Z_MUX2} in EX2 stage.

2.3 Mr. Bruin left out the branch control signal in this HDU_Br design of the 5-stage early branch. Does the CPU still produce the right result? \text{Y/N} Explain.

2.4 Miss Bruin arranged her main memory 8-way LOI (Lower Order Interleaved) for her 80486 processor based system though the block-size for the L1 cache in the processor has a block size of 4 words. There is no L2 cache. What would you like to tell her besides that she should have chosen the 4-way LOI MM?

2.5 In a processor with a 2-way set-associative cache with 1024 sets, there is a LRU (Least Recently Used) bit for every set (total 1024 LRU bits) Due to some VLSI layout error, 10 of these bits were permanently stuck at 1 and surprisingly 10 other LRU bits were permanently stuck at 0. So does it work at all?

2.6 What is the size of the memory address space for each of the two processors below.

1. \text{64-bit data} 24-bit logical address byte-addressable processor: \text{16} Mega Bytes

2. \text{16-bit data} 24-bit logical address byte-addressable processor: \text{16} Mega Bytes
Hints given in that question are
1) /BE7-/BE0 - 8 Bytes so A2, A1, A0 of the address, we got the byte field
2) 16 way lower order interleaving. 16 interleaving needs 4 bits of address. This is the word field, i.e. A6, A5, A4, A3
   now we came to know that one Block has 16 X 8 Bytes (16 words, each word of 8 bytes).
3) 3rd hint is size of the tag ram is 2K = 2 * 2^power(10) = 2^11 so we have 11 set bit set field.
   as Degree Of Set Assiosiativity given is 5.
   the cache size is 2K * 5 * 2^power(7) = 2^10 * 5 * 2^power(7) = 1280 KB
   2^power(7) is taken because it's the block size
   total address = 42 bits
   tag size = 42- (11 bit set field + 4 bit word field + 3 bit byte field) = 24
   TAG RAM width and TAG comparator width should include a valid bit. So it is 24-bit Tag + 1 valid = 25 bits

3 (36 points) 20 min. Cache and Main Memory Organization:

A 64-bit data (D63-D0) 42-bit (logical) address byte-addressable processor (address pins: A41-
A3, /BE7-/BE0) has its cache and MM organized as shown below. Fill in the 9 boxes.

Note: It is common for 64-bit processors to have a 48-bit address but no one populates such a big address space (2^48 bytes). In our examples, for simplicity, we show the MM as fully populated.

3.1 Block size (based on degree of lower-order interleaving of the MM) = 16 Words = 128 Bytes
The design uses _set associative_ (set-associative / direct) mapping. If set-associative, degree of set associativity = 5 blocks/set
The processor address space = 4096 GBytes. Cache size = 128.0 KBytes

3.2 Please divide the address below into appropriate fields and name the fields.

24 | 11 | 4 | 3
---|---|---|---
TAG | SET | Word | Byte
3  ( 36 points) 20 min. Cache and Main Memory Organization:

A 64-bit data (D63-D0), 42-bit (logical) address byte-addressable processor (address pins: A41-A3, /BE7-/BE0) has its cache and MM organized as shown below. Fill-in the 9 boxes.

Note: It is common for 64-bit processors to have a 48-bit address but no one populates such a big address space (2**48 bytes). In our examples, for simplicity, we show the MM as fully populated.

2K TAGs = 2**11 sets = 11-bit set field
=2**11 Block Frames in a Data RAM = [2**11]*16 words in a Data RAM => [2**11]*16 Bytes in a Byte wide bank in the Data RAM = 32 KB

2.1 Block size (based on degree of lower-order interleaving of the MM) = _______ Words = _______ Bytes

The design uses ______________________ (set-associative / direct) mapping.

If set-associative, degree of set associativity = _______ blocks/set

The processor address space = _______ GBytes. Cache size = ________ KBytes

2.2 Please divide the address below into appropriate fields and name the fields.

It is not difficult to get an A in EE457. You need to work for it and seek help from the 457 teaching team on whatever you do not understand. We are eager to help you.

The final topics, virtual memory, exceptions, branch prediction, out-of-order execution, chip multi-threading, chip multiprocessing, cache coherency, locks and mutual exclusion are interesting and challenging too. They are the focus of 50% of the final exam. Best wishes!