## DDCA final exam

Der Test war glücklicherweise leichter als ich es mir vorgestellt hatte. Die Theoriefragen machten (wenn ich mich recht erinnere) 11 Punkte aus und wurden anscheinend erst nachher bewertet. Es ist eine multiple choice Frage gekommen (branch delay slot). Dann eine Frage, welche Operationen länger dauern, Addition, Schiften mit or verknüpft und Division, und eine Frage wozu das Forward gut ist, und man solle es in die Zeichnung einzeichnen mit allen inputs und outputs. (Also schauts euch meiner Meinung nach das fwd und das ctrl an).

Dann 2 Praxisbeispiel mit jeweils entweder 11 oder 12 punkte. Das erste war bei mir eine extrem simple ALU mit ohne alles. :D Kein Zero Flag, kein Carry. Die Operationen waren NOT, AND, OR und XOR, also alles mit VHDL direkt machbar. Weils alles logische Funktionen waren benötigte man auch kein signed oder eine Umwandlung (kommt ja nur bei variablen). Das vhd der ALU war from scratch zu schreiben. Man hatte 2 ausgedruckte Zetteln daliegen auf denen alles mögliche zur Syntax steht. Also wie entity zu schreiben ist usw. Bei mir hat der Zettel nicht viel geholfen, ich empfehle die yntax nochmal anzuschauen. Man kann aber auch schummeln und sich die Angabe zum 2. Beispiel aufmachen, da gibts schon fertige vhd Files (auch eine Testbench) und da kann man einfach rauskopieren. Bin ein bisserl spät draufgekommen.... :D

Ja und eine Testbench musste auch zur ALU erstellt werden. Wieder wie gesagt hier beim 2er Beispiel nachschauen, wie da genau die Syntax geht und man erspart sich einiges. Projektfiles gibt es KEINE. Man muss alles selber bauen. Ich hab Quartus gar nicht erst gestartet, weil ich dafür ja sämtliche Daten zum Chip und so bräuchte, also direkt vom Texteditor ins Modelsim, kompiliert und Fehlersuche betrieben mehr fällt mir zum Beispiel 1 nicht ein. Ich weiß halt nicht ob jeder ne ALU bekommt, aber naja kA ich hoffe mal das beste für euch.

Das zweite Beispiel ist kniffliger auf den ersten Blick. Es geht um "pipelining". Im Prinzip steht viel bla bla vor dem man Angst hat, aber wenn mans mal raus hat, wird man schnell merken, dass da fast nix zu tun ist. Gegeben sind 4 Eingänge und 1 Ausgang. In dem Ding drinnen sind 3 Module. Das können verschiedene sein (glaub ich zumindest, da in meinem src Folder auch noch andere vhd Files waren, nicht verwirren lassen!).

Bei mir war es in etwa so:

a geht auf ein Kastl x1 (ich glaub ein Inverter war das oder so), b geht mit Resultat von a in einen multiplizierer. c und d gehen in einen Adder und das Ergebnis auch in den Multiplizierer. Hört sich komisch an, kurz überlegen, dann sollte es klar sein. Das is wiegesagt nur ein Beispiel, ihr werdet sicher ne andere Konstellation bekommen.

Das wichtige kommt jetzt aber:

Man muss am Eingang und am Ausgang jeweils die Eingange und die Ausgänge (bzw den einzelnen Ausgang) in "ein Register" schieben. Also kurz den Wert halten. In der Angabe ist immer von einem Register die Rede, doch das ist trivialst in vhdl wenn man sich das überlegt. Malediglich die Eingängen muss ja und den Ausgang jeweils immer mit zb der steigenden Clockflanke übernehmen. (auf ein internes Signal, und der Rest wird ja alles via den Components verbunden - und "es rennt scho").

Im Prinzip hatte ich das alles. LEIDER hatte ich das Prinzip es Pipelinings aber verfehlt, und ehrlichgesagt nicht wirklihc erkannt in dem Beispiel (keine Sorge steht eh in der Angabe, ich war einfach nur zu blöd dafür). Ich hätte noch ein Register quasi "in der Mitte" einfügen müssen. Also vorm Multiplizierer und nach den beiden ersten Modulen die die EIngänge bearbeitet haben. Ich habs bei diesem Trivialbeispiel nicht wirklich sehen können warum ich das machen sollte.

Lange Rede, gar kein Sinn: In der Mitte muss nochmal ge-latcht werden weil sonst ists ka Pipeline. "punkt". Im Endeffekt hat er mir dann mit richtiger Simulation und allem nur 5 von 12 Punkte gegeben. Ich habs verkraften können....

Stefan Seifried: Um das ganze besser lesbar zu machen hab ich meinen Senf blau eingefärbt, macht evt. Diskussionen leichter lesbar. Sollten wir das Dokument in einer Art finalen Version veröffentlichen kann mans ja wieder zurückfärben.

Noch ein Mitleser

Weiterer Mitleser (mir gingen die Farben aus, also Background colour)

Alex S.

DDCA Theory composition:

1. A general comparison between two values for less-than/greater-than requires a subtraction and an appropriate evaluation of the most significant bits of the result. How many logic elements does the critical path for a comparison of two n-bit values contain asymptotically (i.e., O(n2 ), O(n), O(log n), . . . )? What about comparing for equality/inequality? Why is a comparison for less-than-zero cheap when using a two’s complement representation?

Answer:

Bei Prüfung auf Gleichheit/Ungleichheit würd ich mal sagen: Man verknüft alle Bits der beiden Werte der Reihe nach mit XOR. Wenn überall 0 rauskommt (alle Bits gleich) - kann zB mit ODER-Verknüpfung aller XOR-Ausgänge festgestellt werden - sind beide Werte gleich, ansonsten ungleich. => O(n) (oder O(n+1) wegen dem ODER)

Ganz blöd: Wäre hier nicht O(1) richtig? Ich meine ich werd doch das XOR parallel auf alle bits ausführen oder?

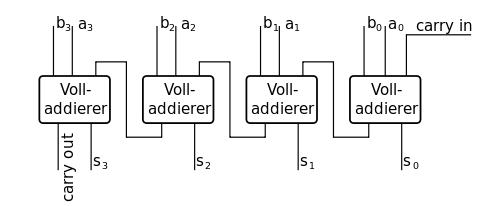
Ja, hab das mit der Gesamtanzahl der benötigten Gatter verwechselt - im kritischen Pfad sollte nur 1 XOR sein.

Wäre es nicht auch möglich, die Prüfung auf Gleichheit/Ungleichheit ebenfalls mit einer Subtraktion (besser gesagt mit der Add. des 2er Komplements) zu lösen?

In diesem Fall würde ich dann auf O(n) kommen.

XOR ist bei Gleichheitsprüfung IMHO einfach nur eleganter.

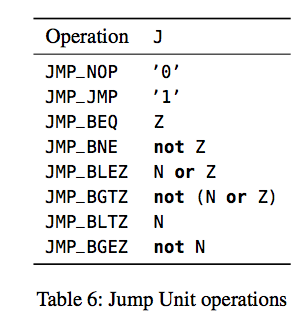
Peter: Denke auch, dass nur 1 XOR im kritischen Pfad ist. Das Ergebnis muss dann noch disjunktiv verknüpft werden, bei OR-Gliedern mit m Eingängen würde das log\_m(n) OR-Stufen entsprechen => Gesamt: 1 + log\_m(n) = O(log(n)) für Prüfung auf Gleichheit/Ungleichheit.

“Why is a comparison for less-than-zero cheap”: Ganz klar, da brauchen wir nur das MSB überprüfen.  
  
“less-than/greater-than”:   
Hab mir das jetzt auch mal grob überlegt und bin jetzt mal bei O(n) gelandet. Ohne mir das Ganze bis runter auf die Gatter-Ebene zu überlegen habe ich mir Folgendes gedacht:  
Die Auswertung des MSB’s ist unabhängig von der Anzahl der Bits und somit konstant. Eine Subtraktion erfolgt in dem man einen der beiden Werte im Wesentlichen invertiert und dann in einen Addierer schickt. Ein Addierwerk kann man eigentlich mit linearem Aufwand erweitern.  
  
Somit käme ich quasi auf O(konstant) für die MSB hab hier nicht zu erschöpfend erklärt).Auswertung + O(konstant) für die nötigen Inverter und nochmals + O(n) für die Erweiterung des Addierwerks. Nachdem ich die konstanten Teile für große n vernachlässigen kann bleibt nur noch O(n) übrig. (hoffe ich

Ich hab hier was gefunden zum Thema Digitaler Komparator.

<http://www.electronics-tutorials.ws/combination/comb_8.html>

1. Table 6 does not contain operations for all boolean combinations of N and Z. Would an operation for N and Z make sense? If so, what would be the high-level comparison? If not, explain why.



Answer:

N und Z gemeinsam macht meiner Meinung nach keinen Sinn. N bedeutet ja dass das Ergebnis negativ und Z bedeutet dass das Ergebnis ‘0’ ist. N und Z würde demnach ‘-0’ ergeben. Dafür fällt zumindest mir keine sinnvolle Branchbedingung ein.

ulli: n und z sind ja nie gleichzeitig gesetzt (n ist eig nur das sign bit (msb) vom ergebnis, wenn das ergebnis 0 dann ist das signbit auch null). daher ist das ergebnis immer 0 und nix passiert.

Beim 2er Komplement kann der fall nicht einträten (es gibt kein -0)

1. The memory unit uses big-endian addressing, where the most significant byte of a word is stored at the lowest address. After storing the word 0x12567348 at address 4, what value should be returned when loading a byte from address 5? Which value should be returned for the half-word at address 6?

Answer :

Byte von Addresse 5 müsste demnach 0x34 sein  
Half-Word von Adresse 6 müsste 0x5678 sein

Ich seh das anders, wenn das MSB die niedrigste Adresse ist, dann sollte es doch so im Speicher liegen: (2 Ziffern sind ein Byte)

Addr.: 7 6 5 4

Daten: 48 73 56 12

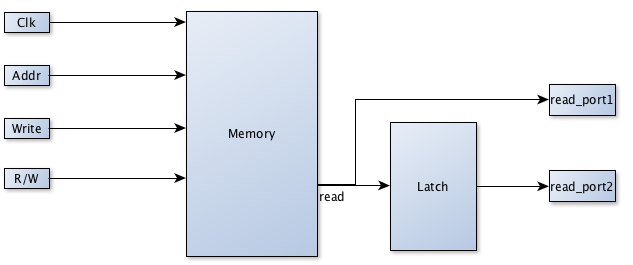
Dann müsste doch beim Laden eines Bytes von addr. 5: 0x56

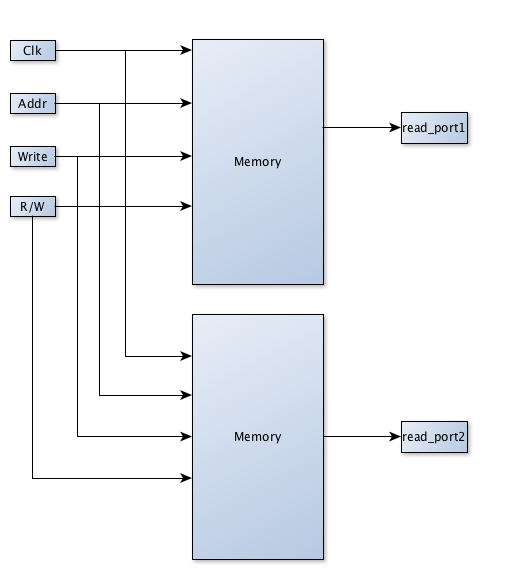
und beim Laden eines HW von addr. 6: 0x7348 zurück kommen, oder??

*Ja, seh ich auch so.*

1. Given memory blocks with one write- and one read-port, how can a memory with one write- and two read-ports be implemented efficiently? What is the overhead, compared to a memory with one write- and one read-port?

Answer:

Wenn ich grad einen read port habe dann muss ich wohl oder übel einmal puffern. Nachteil ist dass ich zwei Clock Cycles brauche und noch ein zusätzliches Latch dass mir das Ergebniss vom ersten Clk vorhält.

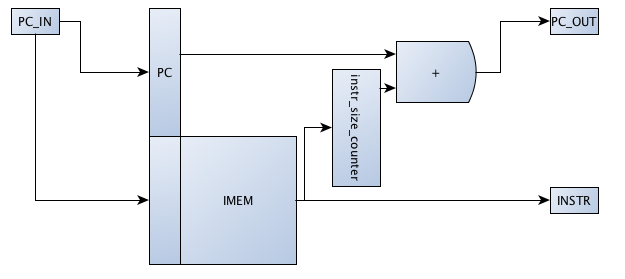


Eine weitere Alternative wäre auch einfach einen zweiten Memory Block einzubauen der Dual beschrieben wird. Vorteil: Alles in einem Clk Cycle, Nachteil viel größerer Platzbedarf am FPGA. PS: Die Skizze ist nicht ganz richtig, damit der zweite read port sinn macht muss natürlich eine andere Addresse möglich sein, sproch die Addresse wird nicht einfach vom ersten Memory zum zweiten durchgeschleust sondern kann im Lese Fall separat gesetzt werden.  
*Wurde in HW-MOD so erklärt, es fehlt aber noch der zweite Addr. Input.*

1. Sketch a fetch stage with variable-length instructions, where the value for the next program counter depends on the instruction that is currently fetched. Which sub-components would be on the critical path in such a fetch stage?

Answer:

subcomponents: imem - fetch - decode - fetch

Ich würde nur einen Fetch verwenden. Selbst bei Instruktionen variabler Länge muss es zwangsweise eine Obergrenze für die Größe geben. Man kann ja die weitere Pipeline auch nicht so leicht erweitern. Man holt sich also erstmal vom IMEM eine Instruktion maximaler Länge und lässt diese dann von einer Komponente (instr\_size\_counter) die Größe bestimmen. Dies kann z.B. mit einer case-when Abfrage auf den Opcode erfolgen. Das Ergebnis wird dann dem Adder zugeführt und gemeinsam mit der Instruction ausgegeben.

1. Explain why it is beneficial to have source registers in the same position for all instruction formats, and why this is less of an issue for destination registers.

Answer:

VLL: weil man bei fast allen instruktionen src register braucht, aber destination nicht immer (zb branches). bei destination braucht man deshalb eh logik je nach instruction also kann man auch gleich den ort mitverändern.

Peter: Da die Adressen der Source Registers bereits bei der aktiven Flanke am Regfile anliegen müssen (um im gleichen Zyklus ausgelesen werden zu können), müssen sie direkt von der Instruktion aufs Regfile gemappt, und müssen deshalb immer an der gleichen Position in der Instruktion stehen. Zum Ermitteln des Destination Registers hat die Decode einen ganzen Zyklus Zeit, da sie erst in den nächsten Stages gebraucht wird (Writeback, Foreward, …).

1. Explain why it is beneficial to multiplex the operands for a single adder over using several adders and multiplex their results. Does the benefit concern rather the performance or the size of the resulting hardware?

Answer:

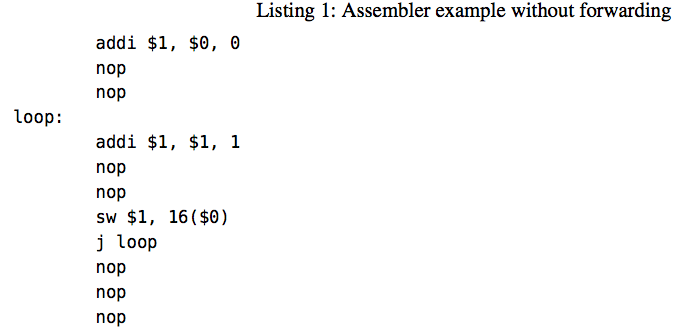
Meiner Meinung nach ist nur die Größe der Hardware davon betroffen, von der Performance her muss man in jedem Fall über einen Multiplexer und einen Adder also bleibts egal. Wenn ich den Multiplexer jedoch hinter den Adder platziere dann brauch ich noch für alle möglichen Kombinationen einen Adder.

Der zweite Teil dieses Satzes macht für mich irgendwie keinen Sinn: “...using several adders and multiplex their results.”

Was würde das bringen? Warum sollte ich mehrere Adder parallel implementieren und mich dann für ein Resultat entscheiden? Pro Instruktion ist ja eh immer nur eine Berechnung durchzuführen. Also außer das es einen Hardwaremehraufwand bedeutet, seh ich da eigentlich keinen Sinn.

Ich glaube der Sinn dahinter soll sein, dass man erkennt das ein Adder weitaus kostspieliger ist als ein Multiplexer. Wenn ich jetzt theoretisch mehrere Adder verwende dann brauche ich die Eingänge dieser nicht zu multiplexen, sondern nur den Ausgang. Im Endeffekt würd ich bei der einen Lösung (n Operationen mit Adder) n\*Adder + 1x Multiplexer(Komplexität n) brauchen. Bei der anderen Lösung 2x Multiplexer(Komplexität n) + 1xAdder.

1. Listing 1 contains seven nop-instructions. How many of these instructions can be removed by reordering instructions, without changing the semantics of the program?



Answer:

addi $1, $0, 0  
 nop

nop

loop:

addi $1, $1, 1

j loop

nop

sw $1, 16($0)

nop

Ich hab mir folgendes dabei überlegt:   
Randbedingungen: Es handelt sich um die Pipeline aus Lvl1 ohne Flush, Forwarding

Somit braucht der jmp 3 Instruktionen bis er ausgeführt wird und das updaten von Werten in den Registern 2 Instruktionen. Somit kann ich nach dem addi in der loop schon mal den jmp Befehl machen und einen Zyklus später das sw ohne irgendwas an der Semantik zu ändern.

Fazit: Einsparung von 3 Instruktionen

mit mind. level2 prozessor:

addi $1, $0, 0  
loop:

addi $1, $1, 1

sw $1, 16($0)

j loop

nop

Frage: Wenn ich den Lvl2 Prozessor hab, kann ich dann nicht auch den Branch Delay Slot ausnutzen und das sw nach dem j loop reinsetzen?

Matthias: Ich bin mir nicht sicher, aber ich glaube ein SW darf nicht im Branch Delay Slot sein. Da wird nämlich die Pipeline gestalled und der Jump bleibt wirkungslos. (Weil die Decode stage die neue instruction nicht übernimmt)

mMn nach ist ein stall egal, der Jump muss mit und ohne stall ausgeführt werden. Problematisch wäre vllt ein Flush???

Stimmt, da hab ich mich wohl vertan.

addi $1, $0, 0  
loop:

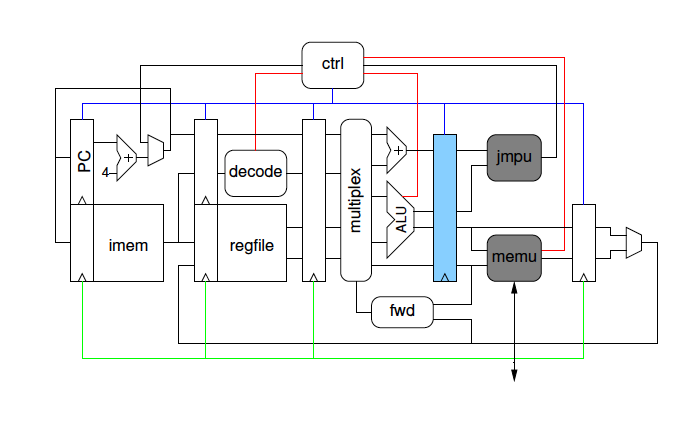
addi $1, $1, 1

j loop

sw $1, 16($0)

1. Explain why forwarding to an instruction immediately after a memory load is infeasible. Where would the critical path be if one would try to forward the result of a memory load to the ALU?

Answer:

Diese Frage ist jetzt meiner Meinung nach etwas knifflig. Ich denke mal man kann hier nicht so einfach gleich das Resultat von der Memory Unit holen, weil man nicht weis wenn die Unit fertig ist. Das Forward das ich seinerzeit implementiert hab war asynchron, es hat also sofort alle Daten weitergeleitet. Wenn jetzt die Memory langsam ist dann kann es sein, dass der richtige Wert erst sehr spät im Taktzyklus bei der ALU ankommt und dann diese die richtige Berechnung nicht mehr rechtzeitig zur clk Flanke am Ausgang hat. (Stichwort: Race Condition) Der kritische Pfad würde sich dadurch verlängern, und zwar von memu über fwd zum multiplex usw., die Memu wäre vorher nicht betroffen gewesen.

1. A branch delay slot is a means to keep the hardware simple while reducing the cost of branches, but may increase the code size. How much is the code size increased with one-cycle branch delay slots, if 15% of the instructions are branches, and 30% of the branch delay slots can be filled by the compiler with useful instructions?

Answer:

Also ohne dass man die Slots mit sinnvollen Instruktionen befüllen kann ist die Codesteigerung 15%. Da man von diesen 15% jedoch 30% befüllen kann bleibt nur eine Steigerung von 70% dieser 15% übrig. Was einem Endergebniss von 10,5% Code size increase entspricht.

X code size with branch delay slots  
x code size without branch delay slots

o overhead produced by delay slotsalc

c size of useful code in delay slots

Y code size with used delay slots

o=15%\*x  
c= 15%\*x\*30%

X=x+o=x\*(1+15%)

Y=X-c=x+o-c=x+15%\*x-15%\*30%\*x=x\*(1+15%-15%\*30%)

=x\*(1+15%-4.5%)=x\*(1+10.5%)  
  
incr=Y/x-1=10.5%

Ich glaub logisch nachdenken bringt mehr als diese Rechnung :D