G-Opgave 2

Michael Thulin, Philip Munksgaard

18. oktober 2010

1 Pipeline-arkitektur

1.1 Implementering af pipelining

Vi tog udgangspunkt i figur 4.60 i COD, og implementerede pipelining i vores kredsløb fra G1. Først opdelte vi vores gamle kredsløb i stages og lavede de forskellige registre til at indeholde data i de forskellige stages.

1.2 Ny control

For at understøtte vores nye stages, samt simplificere kredsløbet og stage-registrene, har vi valgt at konsolidere control-linjerne og samle dem i forhold til de stages hvori de skal bruges. Control-enheden har fire output, sign-extend, jump, link og control-linjen.
Control-linjen består af alle controllinjerne, som så senere bliver delt op og puttet ind i de rigtige trin i pipelinen.

Jump og link-linjerne bruges i j, jal og jr. og er beskrevet nedenfor.

1.3 Forwarding

I vores implementation af forwarding enheden har vi bare fulgt bogens beskrivelser fra side 366 til side 371.

1.4 Hazard detection og stalls

Vi fulgte beskrivelsen af hazard detection på side 372 i vores hazard detector. Når vi detekterer en hazard, får vi en høj på stall linjen. Når stall-linjen er høj, indsætter vi nop i ID/EX trinnet (ved hjælp af en multiplexor) og vi staller IF/ID registret og program counteren.

Bit	Operation	Gruppe
0	AluOP	EX
1	Branch	M
2	MemWrite	M
3	MemRead	M
4	RegWrite	WB
5	MemToReg	WB
6	ALUSrc	EX
7	RegDst	EX
8	AluImmediate0	EX
9	AluImmediate1	EX
10	AluImmediate2	EX
11	AluImmediate3	EX

Tabel 1: Oversigt over bits på control-linjen fra control-enheden

Bit	Betydning
0	ALUOp
1	Branch
2	ALUsrc
3	RegDst
4	ALUImmediate0
5	ALUImmediate1
6	ALUImmediate2
7	ALUImmediate3

Tabel 2: Oversigt over control-bits på EX-trinnet i pipelinen

Bit	Betydning
0	MemWrite
1	MemRead

Tabel 3: Oversigt over control-bits på M-trinnet i pipelinen

1.5 Implementering af j, jal og jr

Vores implementation af j-instruktionen følger bogens eksempel og bruger jump-control linjen fra control-enheden. Når denne linje er høj, springer program counteren og vi staller IF/ID registret.

Jal virker ved at vi springer i ID trinnet og indsætter at vi skal skrive program counter + 4 i register 31 i vores pipeline.

Jr springer i EX-trinnet, og virker ved at vi læser register 31 og springer på samme måde. Vi har ændret i ALUcontrol enheden så vi kan smide en høj værdi på jr-linjen, som springer til den adresse vi har fået ud fra registret.

2 Implementation af primes.c i assembler

Koden kan ses i bilag B.

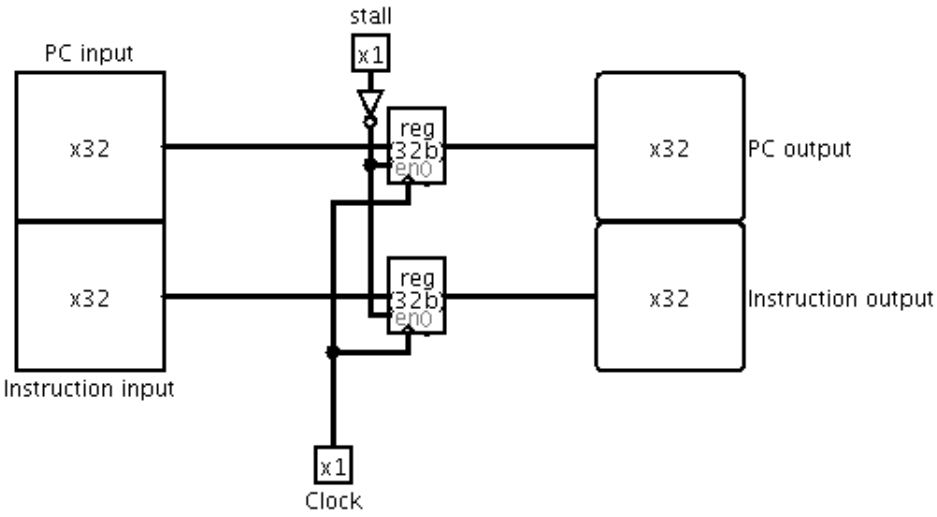
Bit	Betydning
0	RegWrite
1	MemToReg

Tabel 4: Oversigt over control-bits på WB-trinnet i pipelinen

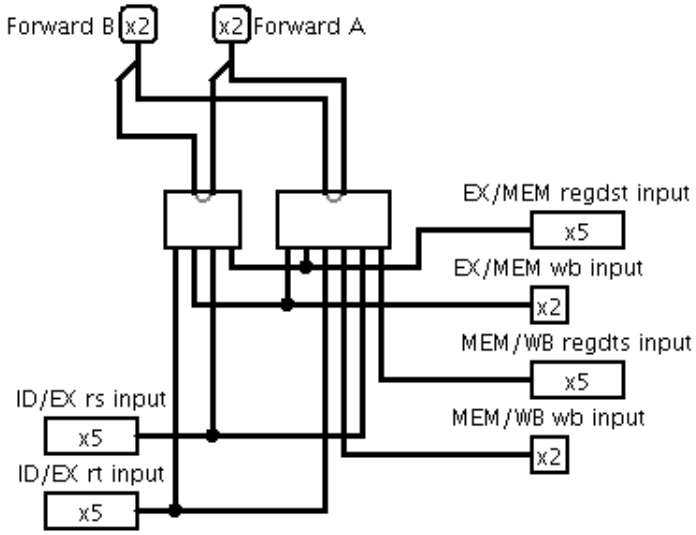
Tabel 5: Oversigt over hvilke registrer der bruges til hvad i primes.asm

Bit	**Betydning**
$s0	i
$s1	p
$s2	idx
$s3	n
$s4	primes

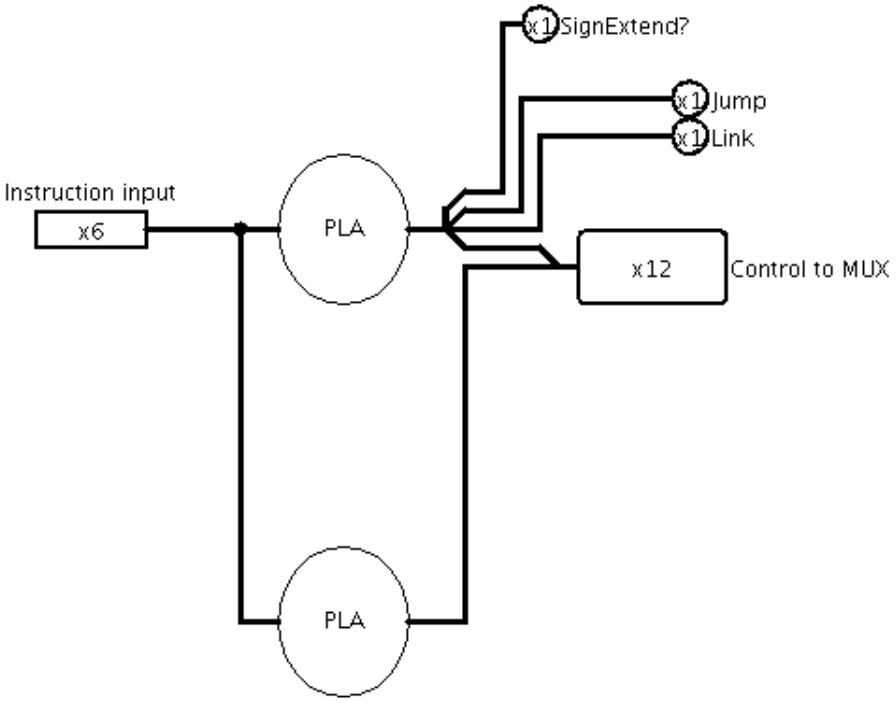
Bilag A: Figurer af udvalgte delkredsløb



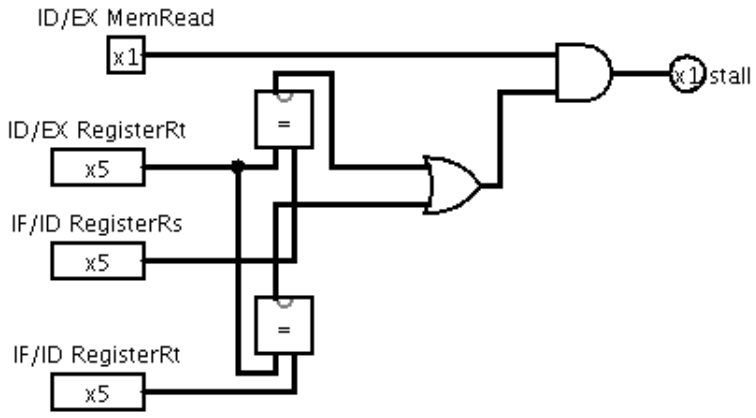
Figur 1: IF/ID registrerne.



Figur 2: Vores forwarding enhed.



Figur 3: Vores control komponent.



Figur 4: Vores hazard detection enhed.

Bilag B: Primes.c i assembler

```
main:   addiu   $a0, $0, 28             #Gem argumentet klar
        jal     largest_prime           # largest_prime(28)

#       addu    $v0, $0, $0             # return 0
        j       end

largest_prime:
        addiu   $sp, $sp, -24           #Giver plads til 4 registre
        sw      $ra, 20($sp)            # Gem retur adressen på stacken
        sw      $s4, 16($sp)            # Gem s-registrene, vi skal bruge s4 til prim
        sw      $s3, 12($sp)            # gem $s3 i stacken, da vi skal bruge s3 til
        sw      $s2, 8($sp)
        sw      $s1, 4($sp)
        sw      $s0, 0($sp)             # vi skal bruge s0 til i

        addu    $s3, $a0, $0            # Gem argumentet n i s3

        addiu   $a1, $0, 4              # Vi vil gange n med 4
        jal     mul                     # for at få [U+FFFD]resen[U+FFFD]dres array
        subu    $sp, $sp, $v0           # Ryk stakpegeren for at give plads til prim
        addu    $s4, $sp, $0            # Gem primes i s4

        addiu   $s0, $s0, 2             # i = 2
L1:     slt     $t0, $s0, $s3           # t0 = i < n?
        beq     $t0, $0, L1_exit        # hvis i >= n hop ud

        addu    $a0, $0, $s0            # find offsetet til primes[i]
        addiu   $a1, $0, 4
        jal     mul
        addu    $t0, $v0, $s4

        sw      $s0, 0($t0)             # gem i i primes[i]

        addiu   $s0, $s0, 1             # i[U+FFFD]g i med 1
        j       L1

L1_exit:
        addiu   $s1, $0, 2              # Initialiser p=2
L2:     addu    $a0, $s1, $0            #Gør klar til at gange p med sig selv
        addu    $a1, $a0, $0
        jal     mul                     # v0 = p*p
        slt     $t0, $v0, $s3           #Sæt[U+FFFD]0 til 1 hvis p*p er mindre end n
        beq     $t0, $0, L2_exit        # Hvis p*p ikke er mindre end n, forlad [U+FFF

        addu    $a0, $s1, $0            #Gør klar til at gange p med 4 (offset ind
        addiu   $a1, $0, 4
        jal     mul                     # v0 = p*4
        addu    $t1, $v0, $s4           # t1 = offset i primes til p. primes + p*4
        lw      $t0, 0($t1)             # t0 = primes[p]
        beq     $t0, $0, L2_continue    # spring videre hvis primes[p] = 0
```

```
        addiu   $s0, $0, 2              # i = 2

W1:     addu    $a0, $s0, $0            # Gør klar til at gange i med p. a0 = i
        addu    $a1, $s1, $0            # a1 = p
        jal     mul                     # v0 = i*p
        addu    $s2, $v0, $0            # idx = i*p

        slt     $t0, $s2, $s3           # t0 = idx < n
        beq     $t0, $0, W1_break       # break hvis idx ikke er < n, altså idx >= n

        addu    $a0, $s2, $0            # gør klar til at gange idx med 4 (offset ind
        addiu   $a1, $0, 4
        jal     mul                     # v0 = p*4
        addu    $t0, $v0, $s4           # t0 = primes + idx*4 = positionen af primes[
        sw      $0, 0($t0)              # primes[idx] = 0

        addiu   $s0, $s0, 1             # i++

        j       W1

W1_break:

L2_continue:
        addiu   $s1, $s1, 1             # inkrementer p
        j       L2

L2_exit:
        addu    $s0, $s3, $0            #  i = n (vi tæller ned fra n og stopper når i
L3:     addiu   $s0, $s0, -1            # i = i - 1
        slti    $t0, $s0, 2             # t0 = i < 2
        addiu   $t1, $0, 1              # t1 = 1
        beq     $t1, $t0, L3_exit       # exit loop hvis i < 2

        addu    $a0, $s0, $0            # Gør klar til at regne i*4 (offset i primes
        addiu   $a1, $0, 4              # a1 = 4
        jal     mul                     # v0 = i*4
        addu    $t1, $v0, $s4           # t1 = i*4 + primes = positionen af primes[i]
        lw      $t0, 0($t1)             # t0 = primes[i]
        beq     $t0, $0, L3             # Hvis primes[i] er nul, loop igen

        j       largest_prime_exit      # return i

L3_exit:
        addu    $s0, $0, $0             # gør returværdien klar

largest_prime_exit:
        addu    $a0, $s3, $0            # Gør klar til at gange n med 4
        addiu   $a1, $0, 4              # Vi vil gange n med 4
        jal     mul                     # for at finde størrelsen af vores array
        addu    $sp, $sp, $v0           # Ryk stakpegeren for at gøre plads til prim

        addu    $v0, $s0, $0            # Gem returværdien

        lw      $ra, 20($sp)            # hent returadressen retur adressen på stack
```

```
        lw      $s4, 16($sp)            # hent s-registrene ind igen
        lw      $s3, 12($sp)
        lw      $s2, 8($sp)
        lw      $s1, 4($sp)
        lw      $s0, 0($sp)
        addiu   $sp, $sp, 24            #G[U+FFFD]lads til 4 registre
        jr      $ra                     # skal fikses

mul:    addiu   $t1, $0, 1              # t1 = 1, vi skal bruge den til at dekremente
        addu    $v0, $0, $0             # returv[U+FFFD]rdien initialiseres til 0

mulloop:
        slti    $t0, $a1, 1             # t0 = b < 1
        beq     $t0, $t1, mulexit       # Hvis b < 1 eller b !> 0, exit
        subu    $a1, $a1, $t1           # dekrementer b
        addu    $v0, $v0, $a0           #L[U+FFFD]gger a til resultatet
        j       mulloop                 # loop

mulexit:
        jr      $ra                     # returner


end:
```