Maskinarkitektur, G4

Daniel Bejder, Anders Bjerg Pedersen
Hold 2

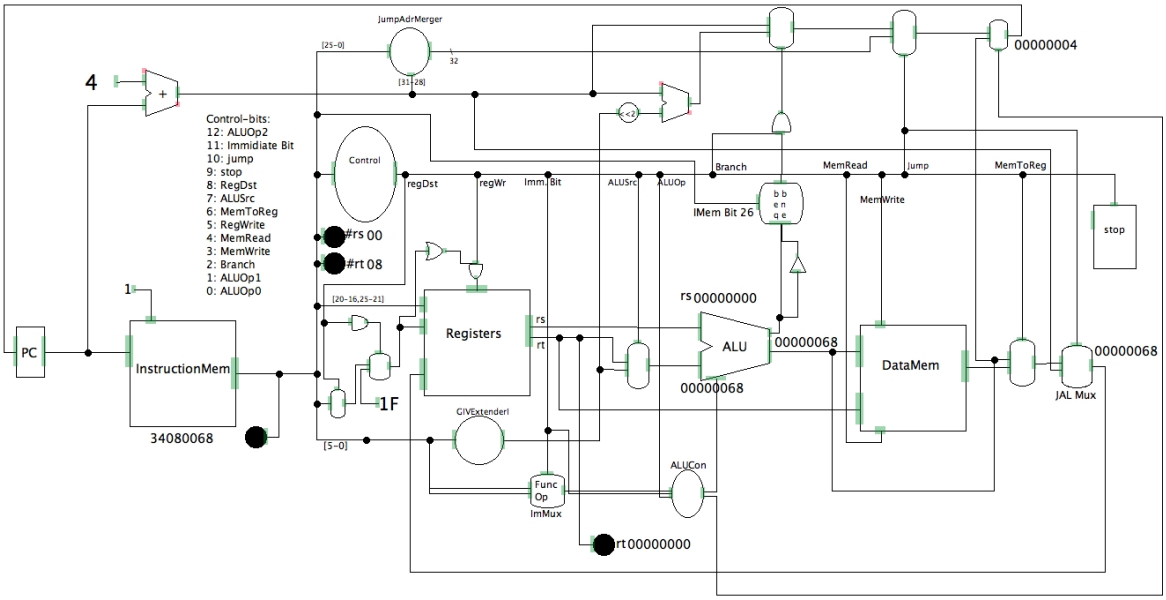
10. januar 2007

Kort om projektet

Med udgangspuinkt i den udleverede (fejlfyldte) XML-fil til G4 har vi konstrueret en 32-bit enkeltcyklus-CPU med understøttelse af funktionerne fra Mini-MIPS og funktionen stop. Der er *ikke* implementeret overflow-kontrol.

Vi har forsøgt så vidt muligt at implementere funktionerne uden at udvide Controlleren mere end højst nødvendigt, kun 1 bit er tilføjet (dog kunne vi grundet en fejl i KredsDesign ikke fjerne en, vi allerede havde tilføjet, ALUOp2). I stedet har vi udvidet kredsløbet med små komponenter.

Figurer over de vigtigste komponenter



Figur 1: Skitse af det overordnede kredsløb

Implementationer

Zero- og SignExtender

Ved immidiate-instruktioner er det vigtigt ikke at ændre på inputværdiers betydning. Når der laves ADDI og SLTI skal der signextendes, mens der ved ANDI og ORI skal zeroextendes for ikke at ændre input.

Vi har lavet en Extender, der tager de 16 bit input fra immidiate-instruktionen samt de 4 mest betydende bit fra OPCODE. Når bittene fra OPCODE er 0010, skal der signextendes, mens der ved 0011 skal zeroextendes. Signextend implementeres ved at sætte bit 15 fra input i bits 16-31 i output, mens til zeroextend bruges bit 19 fra input, der altid er nul, når der skal zeroextendes.

5	4	3	2	1	0	12	11	10	9	8	7	6	5	4	3	2	1	0	Kommentarer
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	R-format (op: 0x0)
1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	lw (op: 0x23)
1	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	sw (op: 0x2b)
0	0	0	1	0	x	0	0	0	0	0	0	0	0	0	0	1	0	1	beq (op: 0x04)
1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	stop (op: 0x3F)
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	jump (op: 0x2)
0	0	1	x	x	x	0	1	0	0	0	1	0	1	0	0	0	1	0	I Format
0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	jal

Figur 2: Control-PLA'en

9	8	7	6	5	4	3	2	1	0	3	2	1	0	Kommentarer
0	0	0	0	x	x	x	x	x	x	0	0	1	0	Lw / Sw (2)
0	0	x	1	x	x	x	x	x	x	0	1	1	0	Branch (6)
x	0	1	x	x	x	x	0	0	0	0	0	1	0	Add (2)
x	0	1	x	x	x	x	0	1	0	0	1	1	0	Sub (6)
x	0	1	x	x	x	x	1	0	0	0	0	0	0	And (0)
x	0	1	x	x	x	x	1	0	1	0	0	0	1	Or (1)
x	0	1	x	x	x	1	0	1	0	0	1	1	1	SLT (7)
0	0	1	x	0	0	1	0	0	0	1	0	0	0	JR ( ) ALUOP3

Figur 3: PLA til ALUControlleren

Immediates

Her lavede vi en immediate-bit fra Control, der vælger mellem register- eller immediateinput til ALU'en. Desuden sender vi for immediates OPCODE ind i ALUControl i stedet for FUNCT.

Denne implementation medfører meget behageligt, at også SLTI virker, så snart SLT virker.

Branches

BEQ var allerede implementeret. For at implementere BNE har vi taget bit 26 fra instruktionen ("branch-bitten) til at skelne mellem beq og bne, som styrer en Mux, der vælger mellem zero- eller not-zerooutputtet fra ALU'en under branches.

SLT

Dette var bare at udfylde den allerede indsatte PLA i ALU'en. Med en linie for hver af de seks muligheder. Se Figur 4.

2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
x	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
x	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figur 4: PLA hørende til SLT i vores ALU

Jumps

Til JAL har vi lavet en Mux, en Driver og en And til at vælge targetregister = 31, i stedet for de 0 i instruktionen. Desuden har vi indsat en Mux (JalMux) efter MemToRegMux, der vælger mellem PC+4 og MemToRegMux'ens output.

Til JR har vi indsat en Mux, der vælger imellem den normale jumpadresse og outputtet fra ALU'en (det loadede register).

Test af CPU

Vi har genereret 3 testfiler (se Bilag 1). *test1.asm* tester immediate- (fraregnet SLTI) og R-typer, *test2.asm* tester slt og slti, test3.asm tester BEQ og BNE. Derudover har vi brugt PrintFib.asm til at teste jump-funktionerne. Til sidst har vi kørt Andresens testfil, *test1.asm* (i vores bilag navngivet AndresenTest). Sidstnævnte er kørt med sammenligning af trace-filen, genereret med MASM, de andre er kørt internt i KredsDesign. Alt ser ud til at virke.

Bilag 1: Kildekode til testprogrammer

test1.asm

```
; TEST 1: R/I-formater
; Require ori,addi,andi,add,and, sw and STOP

Main:
  ori   $8, $0, 104   ; 104 -> $8
  sw    $8, -12($0)   ; print 104 ($8)

  addi  $9, $8, 103   ; 104 + 103 = 207 -> $9
  sw    $9, -12($0)   ; print 207 ($9)

  andi  $3, $8, 26    ; 104 & 26 = 8 -> $3
  sw    $3, -12($0)   ; print 8

  or    $4, $3, $8    ; 8 | 104 = 104 -> $4
  sw    $4, -12($0)   ; print 104 ($4)

  and   $2, $8, $9    ; 104 & 207 = 72 -> $2
  sw    $2, -12($0)   ; print 72 ($2)

  add   $10, $3, $3   ; 8 + 8 = 16 -> $10
  sw    $10, -12($0)  ; print 16 ($10)

stop
```

test2.asm

```
; TEST 2: SLT/SLTI
; Requires slt,slti,ori,sw og STOP

Main:
  ori   $1, $0, -4    ; -4 -> $1
  ori   $2, $0, 17    ; 17 -> $2
  slt   $3, $1, $0    ; 1 -> $3
  sw    $3, -12($0)   ; print 1 ($3)
  slt   $4, $0, $0    ; 0 -> $4
  sw    $4, -12($0)   ; print 0 ($4)
  slt   $5, $2, $1    ; 0 -> $5
  sw    $5, -12($0)   ; print 0 ($5)

  slti  $6, $2, 22    ; 1 -> $6
  sw    $6, -12($0)   ; print 1 ($6)
  slti  $7, $1, -4    ; 0 -> $7
  sw    $7, -12($0)   ; print 0 ($7)
  slti  $8, $1, -2    ; 1 -> $8
  sw    $8, -12($0)   ; print 1 ($8)  $

stop
```

test3.asm

```
; TEST 3: BEQ/BNE
; Requires beq,bne,ori,add,sw and STOP

Main:
  ori   $1, $0, 2     ; 2 -> $1
  ori   $2, $0, 16    ; 16 -> $2
  bne   $1, $0, Bneq  ; jump to Bneq

Quit:
  stop

Bneq:
  add   $3, $1, $1    ; 2 + 2 = 4 -> $3
  sw    $3, -12($0)   ; print 4 ($3)
  bne   $1, $1, Bneq  ; do nothing
  beq   $0, $2, Bneq  ; do nothing

Beq:
  add   $4, $2, $2    ; 16 + 16 = 32 -> $4
  sw    $4, -12($0)   ; print 32 ($4)
  beq   $2, $2, Quit  ; jump to Quit
```

AndresenTest.asm

```
        ori         $1,  $0,   0x8000
        add         $2,  $1,   $1
        ori         $3,  $2,   0x8000
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $2,  $2,   $2
        add          $3,  $2,   $2
        add          $4,  $3,   $3
        add          $5,  $4,   $4
        or         $6,  $2,   $3
        or         $6,  $6,   $4
        or         $7,  $6,   $5

        addi         $1,  $0,   4
        addi         $2,  $0,   6
        addi         $3,  $0,  -4
        addi          $4,  $0,  -6
        slt         $8,  $7,  $6
        slt         $9,  $6,  $7
        slt         $5,  $1,  $2
        slt         $6,  $2,  $1
        slt         $7,  $3,  $4
        slt         $8,  $4,  $3
        slt         $9,  $1,  $3
        slt         $10, $3,  $1

        ori         $2,  $0,   0xFFFF
        andi         $2,  $2,   0x8000
        addi         $3,  $0,   0x8000
        slti         $4,  $1,   0x8001
        slti         $5,  $1,   0x7FFF
        ori         $6,  $0,   0x8000
        andi         $2,  $6,   0x8001
        sw         $0,  -4($0)

        j j_f
        sw         $2, 0($0)
jal_b:
        addi          $31, $31, 4
        jr         $31
        sw         $2, 4($0)

j_b:        beq         $2,$2, beq_f
```

```
        sw        $2, 8($0)

j_f:
        jal jal_b
        sw        $2, 12($0)
        j j_b
        sw        $2, 16($0)

beq_b:       beq        $0,$31, fail
        bne        $0,$0, fail
        bne        $2,$2, fail
        jal        jal_f
        sw        $2, 20($0)

beq_f:       beq        $0, $0, beq_b
        sw        $2, 20($0)

bne_b:       bne        $0,$2, bne_f
        sw        $2, 20($0)

jal_f:       bne        $2,$0, bne_b
        sw        $2, 20($0)

fail:       sw        $2, 20($0)
        stop

bne_f:
        sw        $0,  -4($0)
        stop
```