15 Kritisk sti multiplekser

Z <= (A **and** S) **or** (B **and not** S);

2 porter à 50ps + inverter à 10ps gir 110ps

16 Kritisk sti MUX-2

Splitter hver tre input AND port i 2 gir 8 AND-porter.

4 input OR må bli 2 2input fulgt av en 2 input, så 3 OR-porter.

AND-portene har 2 nivåer, OR portene 2, + inverter ved inngang til AND portene =>

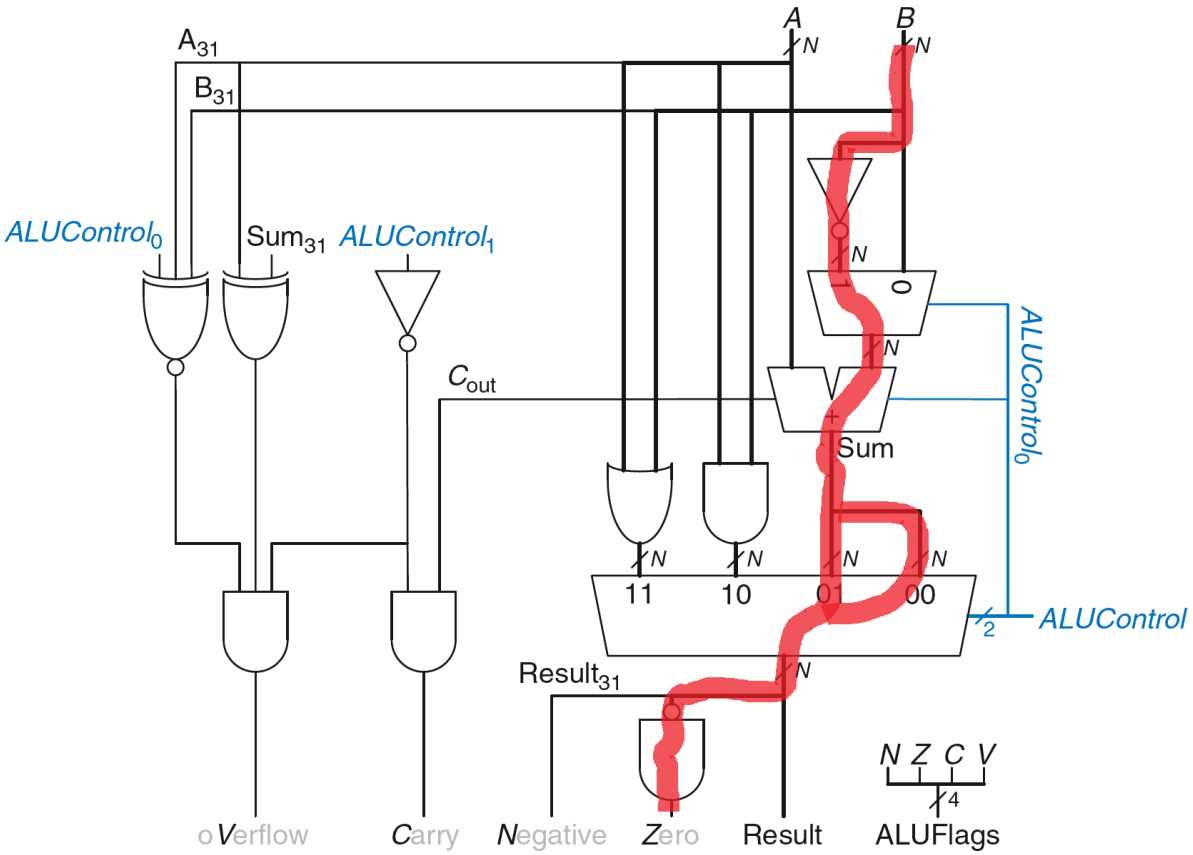
(10+4*50)ps = 210ps

17 Kritisk sti fulladder:

sum <= A **xor** B **xor** Cin;
Cout <= ((A **xor** B) **and** Cin) **or** (A **and** B);

Tre porter, så 150 ps.

18 Kritisk sti ALU-1



B til Z er lengste vei. Man kan argumentere for ALUcontrol0 til Z fordi den inverteres når vi ikke inverterer B (Samme lengde), men det valget ble ikke tilbudt for å ha ett entydig svar, som er lett å forholde seg til ifht kritisk sti.

Kritisk sti, ALU-2

Addisjonskrets fra A til Sum: 3200ps

Fulladder 0, XOR (A,B) : 50ps

Fulladder0-30, (AND, OR) x 31: 100ps*31 = 3100ps

Fulladder31, XOR (Cin, «utgang» A xor B) : 50ps

ALU:

Inverter: 10ps

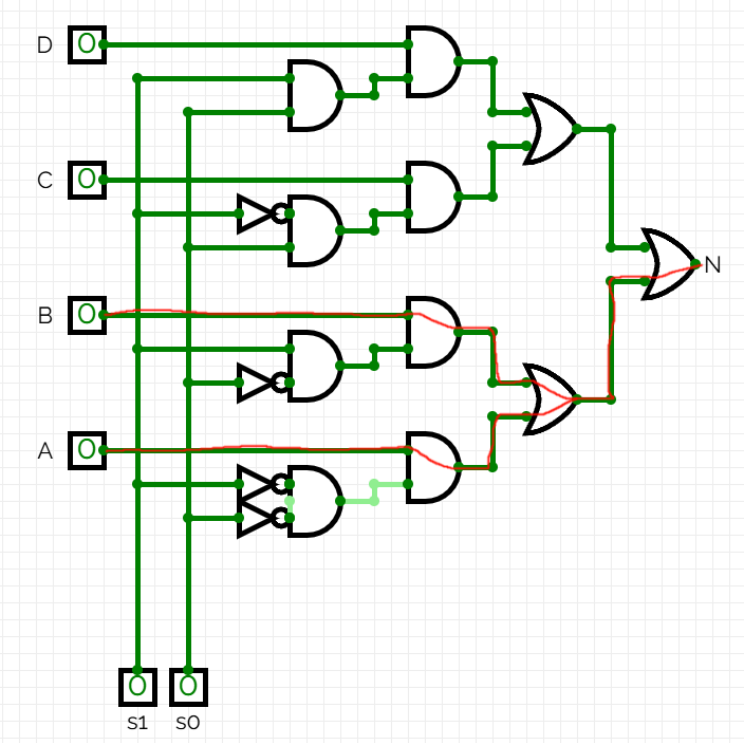
2-input Mux :100ps
AND: 50 ps , OR: 50ps:

Addisjonskrets: 3200ps

4-input Mux: 150ps

Merk: Logikk som bare brukes til kontrollsignal blir ikke en del av kritisk sti, da kontrollsignalet er klart før de øvrige inngangene som følge av øvrig kritisk sti i ALUen.

Organiseringen av AND-portene avgjør om man får 200ps eller 150ps avhengig av hvordan kontrollsignalene rutes versus inngangene (her ABCD).



4-input mux kritisk sti i ALU. Av Leander M.M.Lindgren

Zero : 260ps
$\log_2(32)$ * 50ps = 5*50ps = 250ps
Inverter: 10 ps

HELE: (10 + 100 + 3200 + 150 + 260)ps = 3720ps (3720ps-3770ps godtas).
=> f_{max} = 1/3720ps = 269 MHz

Cache-2 utregning (Alternativ visning av utregning)

2kB = 2048Byte (2^{11})

4 Ord per blokk og 4 byte per ord gir 16 byte per blokk (2^4)

Antall Sett = Kapasitet/blokkstr = $2^{11}/2^4 = 2^7$

a) 128 sett

Leseoperasjoner

2048 = 0x800

2048-1 = 0x7FF

Hvert sett tar adresse bredde 0-F

Så mao

0x010-1F er linjen over 0x000

Minneadresse + 0x80X overskriver Minneadresse,

DVS

0x010,	**0x014,**	**0x01C,**	**0x020,**	**0x820,**	**0x424,**	**0x42C,**	**0x82C**
M(t),	H,	H,	M(t) ,	M(tk),	M(t),	H,	H

b) 4 miss første gang

Merk : 0x820 skriver over 0x020 og vise versa

10x: 1x+ 9x:

H,	H,	H,	M(k),	M(k),	H,	H,	H

c) (4 hit + 9*6hit) /(8*10) = (4+54)/80 = 58/80 = 0,725 treffrate

Med to veis assosiativ og blokkstørrelse på 2 ord, så får vi samme antall sett, men adresserom for cache blir halvparten.

DVS

linje 0 : 0x0000 – 0x0007

linje 1: 0x0008 – 0x000F

linje3: 0x0010- 0x0017, osv

adresse 0x0400 skriver over 0x0000

⇨ Treffrate

Utregning :

0x03FF- 0x0380	0x03FF- 0x0380
…	-0x03F0
	…
	-0x0010
…	0x000F -0x0008
0x0007-0x0000	0x0007-0x0000

m8-0x82C, h2.8	m7-0x42C, h2.7
	…
m5-0x820, mk2.4, mk2.6	m4-0x020, mk6-0x424,mk2.5
	m3-0x01C, +h2.3
	m1-0x010, h2-0x014, h2.1+h2.2

0x010, 0x014, 0x01C, 0x020, 0x820, 0x424, 0x42C, 0x82C

1: M, H, M, M, M, M, M, M

2-10:H, H, H, M, M, M, H, H

(3,4,5 bytter på hvilken side i cache de misser)

1: (1 + 9*5)/8*10 = 46/80 = 0,575

ALU-flagg, utregning

0x4A50 – 0x05A2 =

```
   Binært:
      0100 1010 0101 0000
   -  0000 0101 1010 0010
```

«=»

```
MENTE 11111  1   1 1
       0100 1010 0101 0000
 +     1111 1010 0101 1110
 =   1 0100 0100 1010 1110
```

Vi får carry ut fra addisjonskrets.

Carryut ANDes med *not* $ALUControl_1$ => C = "1"
(ALUControl = 01, så $ALUControl_1$ = 0)

Vi får ikke V fordi A_{31} XOR SUM_{31} er 0 ("0" XOR "0" = "0"), og mater 3-input AND-gate med 0.

Vi får ikke N fordi $Result_{31}$ er 0.

Vi får ikke Z fordi mer enn ett bit er 1

Vi får ikke «ingen» fordi C blir satt.

Merk: *Ikke alle prosessorarkitekturer deler ARM sin implementasjon. Det finnes arkitekturer som ikke setter carry flagg når vi har et positivt resultat etter subtraksjon, men det gir et annet skjema.*