G1
Maskinarkitektur
Efterår 2011

Jens Fredskov
Naja Mottelson
Søren Pilgård

19. september 2011

Indhold

1 Indledning

Nærværende rapport tjener som dokumentation af gruppens arbejde med første godkendelsesopgave. Den indeholder korrekte og afprøvede løsninger på samtlige underopgaver.

2 g1-1

På Figur 2 ses vores implementation af de grundlæggende logiske gates (AND, OR, NOT, XOR) vha. NAND-gates. Som det ses følger vores implementering metoden i lærebogens Appendix C.

3 g1-2

Overordnet er vores 4-bit ALU (se Figur 5) implementeret som en serie af fire 1-bit ALU'er (se Figur 4) efter samme logik som den 32-bit ALU der præsenteres i Appendix C-29. Her forbindes CarryOut-outputtet fra de mindre betydende bits til CarryIn-inputtet for de mere betydende. Nedenfor er en gennemgang af de operationer 1-bit ALU'en understøtter og deres implementering:

AND, OR ALU'ens grundlæggende logiske funktioner benytter de indbyggede gates i logisim.

NOR Outputtet til NOR udregnes som NOT A AND NOT B.

Addition ALU'en benytter et Adder-modul (se Figur 6), som vi i gruppen har implementeret vha. logisims Combinatorial Analysis-værktøj og sandhedstabellen i Figur C.53.

Subtraktion Substraktionsfunktionen benytter samme Adder, blot med én af operanderne inverteret.

Set on less than Set on less than-operationen (SLT) er en komposit operation: Først sammenlignes A og B, hvilket giver resultatet 1 hvis A $<$ B, 0 ellers. Herefter sættes alle inputtets bits til 0, med undtagelse af mindst betydende bit som sættes til resultatet af sammenligningen. Sammenligningen udfører vi tage differencen på de to input, eftersom en negativ difference $=> A < B$. Nedenfor beskrives logikken bag bla. SLT-implementationen nærmere.

Som det ses adskiller seriens sidste 1-bit ALU sig fra de foregående ved at understøtte yderligere funktionalitet til at undersøge for overløb (se Figur 7) samt håndtering af SLT-operationen. Denne sidste del af logikken har vi implementeret efter samme algoritme som beskrives i Appendix C-31-C-35. Bogens implementering benytter sign-bitten fra adderen som output til SLT-operationen, hvilket undlader at tage højde for over- og underløb ved to-komplementsaritmetik. Vi håndterer dette med en XOR-gate imellem outputtet fra overløbsmodulet og adderen.

Den færdige 4-bit ALU adskiller sig fra de forskellige 1-bits ALU'er ved også at give outputtet ZERO. Dette beregnes ved at føre outputtet fra samtlige operationer igennem en XOR-gate.

4 g1-3

Vores 32-bit skifteenhed er implementeret som et enkelt kredsløb bestående af fem 4-IN multiplexere. Hver multiplexer er forbundet til en enkelt bit i shamt og vil, hvis pågældende bit er sat, skifte inputtet hhv. 1, 2, 4, 8 og 16 pladser. Hvilken skifteoperationen der benyttes specificeres af multiplexerens op-input. Hvis bitten i shamt er lav videresendes inputtet uændret. [NB! Det med det fjerde MUX-input!]

Skifteoperationerne selv er implementeret ved brug af logisims sign extender-modul således at der kopieres et antal bits (hhv. 1, 2, 4, 8 og 16) ind i inputtet. Ved udførelse af sll-operationen erstatter sign extenderens output de bagerste n bits i inputtet, hvor srl-operationen erstatter de forreste n bits. Ved udførelse af sra-operationen indkopieres inputtets mest betydende bit - dette har vi implementeret vha. en AND-gate imellem den mest betydende bit og op-inputtet. Som det ses understøtter vores skifteenhed yderligere operationen shift left arithmetic (sla), implementeret på samme vis som sra.

En anden måde at implementere skifteoperationerne ville være at splitte wires imellem et 32-bit input og output manuelt. På denne måde ville man være nødsaget til at implementere en multiplexer for hver

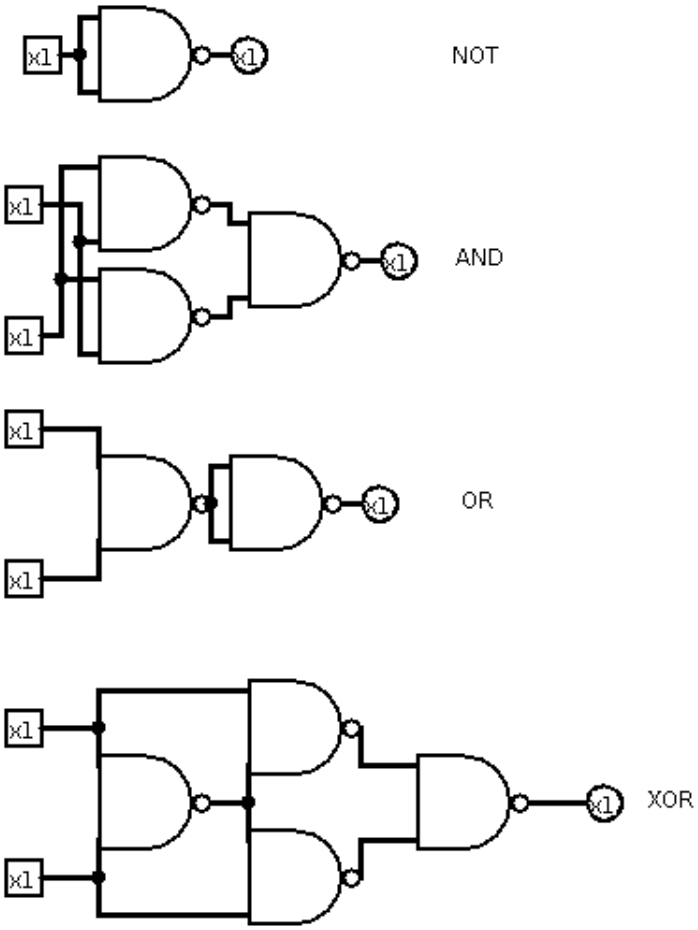
NB! Mangler argumentation for korrekthed/afprøvning! For at afprøve skifteenheden lavede vi et lille testkredsløb der fodrede skifteenheden med tilfældige data. Vi sammenlignede så resultaterne med input værdierne ved simpelt at opskrive dem over hinanden forskudt med shamtværdien. På denne måde kunne vi let teste skifteenheden med en stor række data. I 1 ses et udsnit af de værdier testkredsløbet kørte, det ses let at værdierne er skiftet korrekt.

op	shamt	in	result
00	00000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000
00	10000	10100001 01000111 01000010 11010000	01000010 11010000 00000000 00000000
11	00100	10100001 00101010 00000101 00100100	11111010 00010010 10100000 01010010
10	10101	10100001 00100010 01011100 11010101	00000000 00000000 00000101 00001001
10	11100	10100001 00111101 11000110 10111100	00000000 00000000 00000000 00001010
11	10110	10100001 00111110 00110111 00010110	11111111 11111111 11111110 10000100
11	00011	10100001 00111001 01100101 10000011	11110100 00100111 00101100 10110000
00	00110	10100001 00001111 00011011 11100110	01000011 11000110 11111001 10000000
01	11010	10100001 00011111 00000000 10110100	11010011 11111111 11111111 11111111

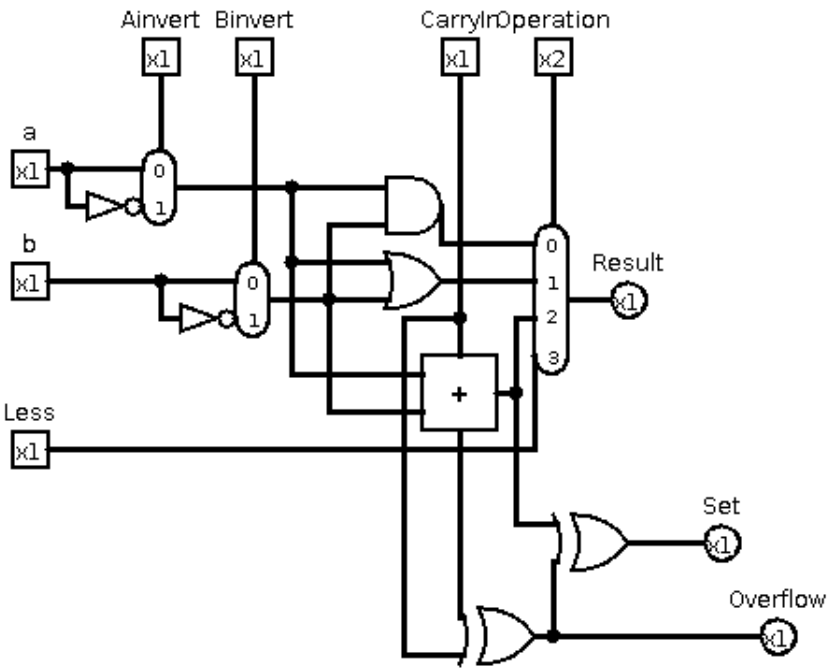
Figur 1: Et udsnit af testkredsløbets værdier

5 Appendix

5.1 NOT-, AND-, OR- og XOR-gates



Figur 2: NOT-, AND-, OR- og XOR-gates af NAND-gates



Figur 3: 1-bit ALU m. undersøgelse af overløb

5.2 ALU, 1-bit m. overflow

5.3 ALU, 1-bit u. overflow

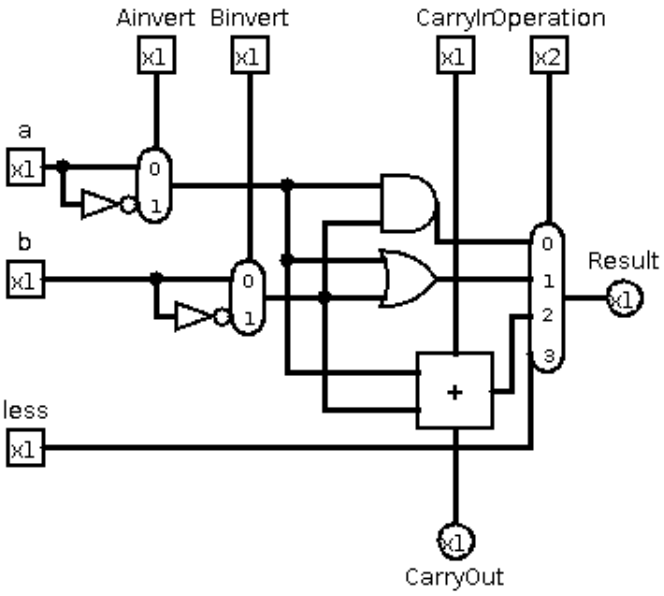
5.4 ALU, 4-bit

5.5 Adder

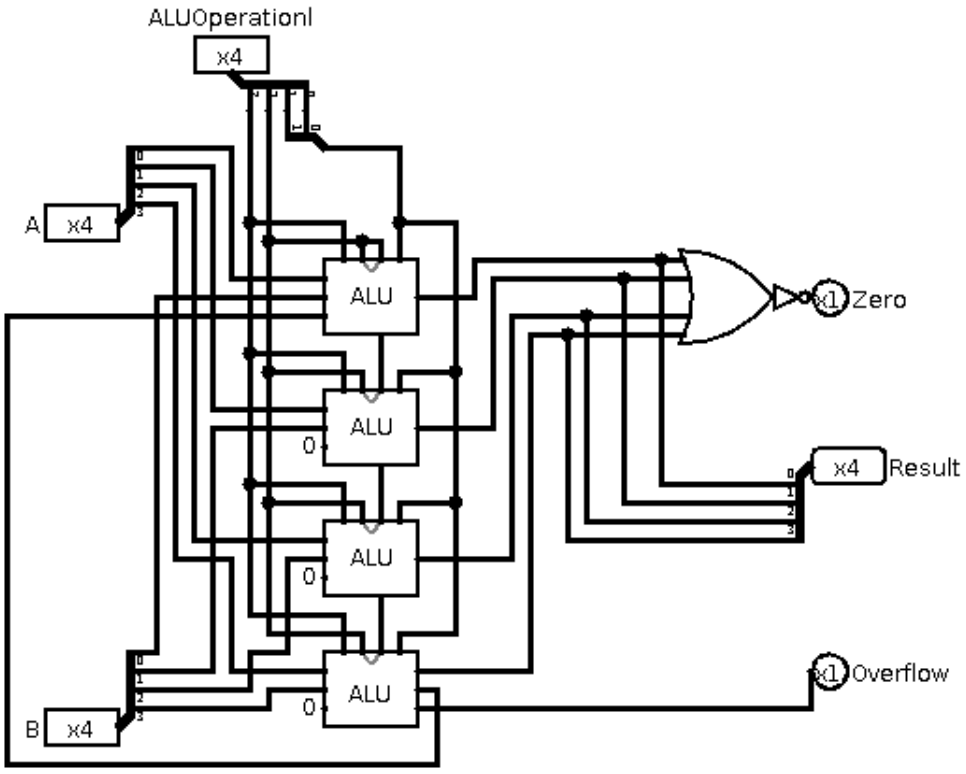
5.6 Overløbskontrol

5.7 Multiplexere

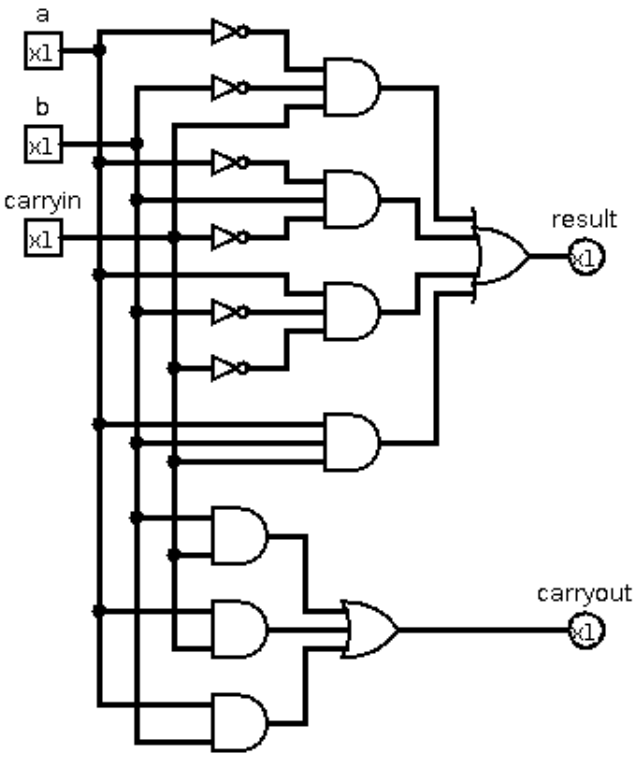
5.8 32-bit skifteenhed



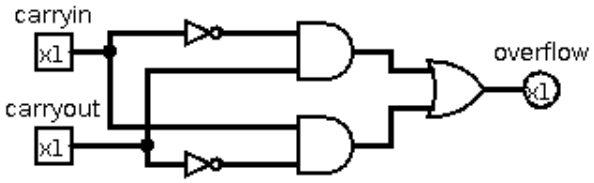
Figur 4: 1-bit ALU u. undesøgelse af overløb



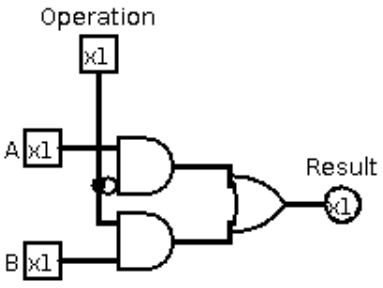
Figur 5: ALU, 4-bit



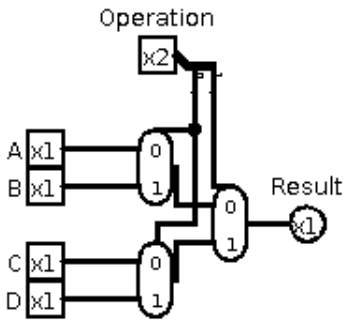
Figur 6: 1-bit adder



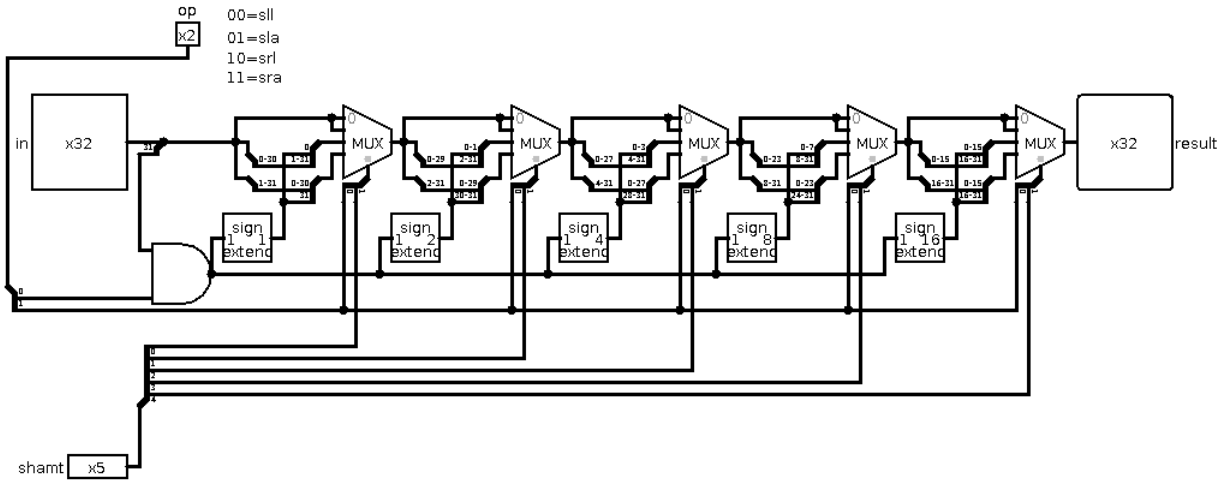
Figur 7: Logikken til undersøgelse af overløb



Figur 8: MUX, 2 bit IN



Figur 9: MUX, 4 bit IN



Figur 10: 32-bit skifteenhed