

Grafitande Räknare - Designskiss

Hannes Haglund hanha265

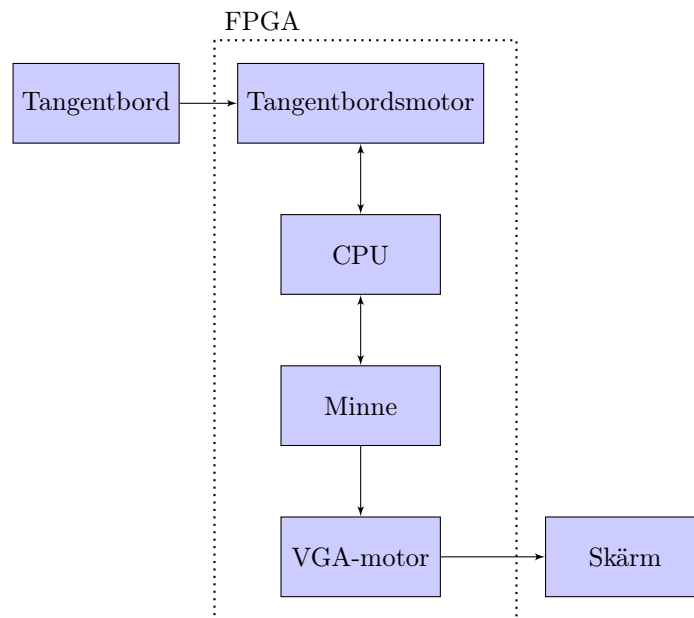
Felix Härnström felha423

Silas Lenz stille914

21 mars 2016

1

Vi ska implementera en generell dator av mikroprogrammerad, ej pipelinead typ. Mjukvaran skrivs i assembler. Vi har en VGA-motor, en tangentbordsavkodare och möjligtvis touchavkodare samt motor för dess skärm (utökningsmål).



1.1 CPU

Vår processor är mikroprogrammerad, med delat data och programminne. Vi använder 32-bitars ordbredd. Endast blockram används.

Processor laddas alltid med samma program vid start.

Nästan allt arbete utförs av processorn, förutom tangentbordsläsning, och utritning av bildminnets innehåll. Dessa uppgifter inkluderar: beräkningar, historik, parsing av input, beräkning av graf, och så vidare.

1.1.1 Instruktioner

Vi har följande adresseringsmoder:

- Direkt
- Omedelbar
- Indirekt

Följande instruktionsmängd:

- | | | | | |
|--------|---------|-------|---------|--------|
| • ADD | • MULTF | • ASR | • BRA | • HALT |
| • ADDF | • DIVF | • BEQ | • BRF | |
| • SUB | • AND | • BMI | • LOAD | |
| • SUBF | • ASL | • BNE | • STORE | |

En utförlig förklaring till varje instruktions argument, resultat, adresseringsmoder, och påverkade flaggor finns tillgängligt som bilaga, på engelska.

1.2 Grafik

Vi delar upp vår display i två kolumner, där ena hälften använder tiles och andra hälften använder en bitmap i svartvitt. Räknaren (text) använder sidan med tiles, och grafen använder bitmapsidan.

Upplösning 640x480. Både tiles och bitmap i svartvitt. Uppdateringsfrekvens 60Hz.

Processorn skriver tilenummer samt bitmapen direkt till bildminnet, utan att synkronisera med bilduppritningen.

1.3 I/O

Input via PS/2 med en avkodare i VHDL. Avkodaren skriver ett tecken till en egen minnesplats som kan läsas av processorn via STORE. Vi låter instruktionen ta en virtuell adress som argument, och en viss adress som överskrider processorns minnesstorlek får referera till avkodarens minnescell.

Via en synkron *read_confirm*-signal så berättar processorn för avkodaren att den lyckats läsa ett tecken, varpå värdet på minnesplatsen nollställs och avkodaren påbörjar läsning av nästa tecken.

Hämtad input ritas ut i ett konsolfönster på skärmen, och interpreteras vid nedslag av returknappen. Tal matas in i form av flyttal (separerad med punkt), och uttryck skrivs i reverse-polish-notation.

1.4 Minne

Vi har följande minnen:

- PC (rw)
- ASR (rw)
- IR (rw)
- μ PC (rw)
- μ Minne (rw)
- Programminne (rw)
- 6 generella dataregister (rw)
- Statusregister (r)
- Bildminne

Med ordbredden 32 bitar.

1.5 Programmering

Vi skriver en assembler, med lite syntaktiskt socker för loopar och if-satser.

1.6 Milstolpe

En fungerande processor som kan rita ut flyttal från en adress i minnet med hjälp av VGA-motor.

Bilaga A Instruktionsuppsättning

ADD	Add signed integers
Operation:	$[\text{destination}] \leftarrow [\text{source}] + [\text{destination}]$
Syntax:	ADD <ea>,Dn ADD Dn,<ea>
Description:	Add the source operand to the destination operand and store the result in the destination location.
Condition codes:	X N Z V C * * * * *

The X-bit and C-bit are both set if carry is generated. The N-bit is set if the sum is negative. The Z-bit is set if the sum is zero. The V-bit is set if overflow occurs (in which case the Z-bit and the N-bit are undefined).

ADDF	Add signed floating-point numbers
Operation:	$[\text{destination}] \leftarrow [\text{source}] + [\text{destination}]$
Syntax:	ADDF <ea>,Dn ADDF Dn,<ea>
Description:	Add the source operand to the destination operand and store the result in the destination location, interpreting operands and sum as signed floating-point numbers.
Condition codes:	X N Z V C * * * * *

The X-bit and C-bit are both set if carry is generated. The N-bit is set if the sum is negative. The Z-bit is set if the sum is zero. The V-bit is set if overflow occurs (in which case the Z-bit and the N-bit are undefined).

SUB	Subtract signed integers
Operation:	$[\text{destination}] \leftarrow [\text{source}] - [\text{destination}]$
Syntax:	SUB <ea>,Dn SUB Dn,<ea>
Description:	Subtract the destination operand from the source operand and store the result in the destination location.
Condition codes:	X N Z V C * * * * *
	The X-bit and C-bit are both set if carry is generated. The N-bit is set if the difference is negative. The Z-bit is set if the difference is zero. The V-bit is set if overflow occurs (in which case the Z-bit and the N-bit are undefined).

SUBF	Subtract signed floating-point numbers
Operation:	$[\text{destination}] \leftarrow [\text{source}] - [\text{destination}]$
Syntax:	SUBF <ea>,Dn SUBF Dn,<ea>
Description:	Subtract the destination operand from the source operand and store the result in the destination location, interpreting operands and difference as signed floating-point numbers.
Condition codes:	X N Z V C * * * * *
	The X-bit and C-bit are both set if carry is generated. The N-bit is set if the difference is negative. The Z-bit is set if the difference is zero. The V-bit is set if overflow occurs (in which case the Z-bit and the N-bit are undefined).

DIVF	Signed floating-point divide
Operation:	$[\text{destination}] \leftarrow [\text{destination}] / [\text{source}]$
Syntax:	DIVF <ea>,Dn
Description:	Divide the destination operand by the source operand and store the result in the destination, interpreting operands and result as signed floating-point numbers.
Condition codes:	X N Z V C - * * * 0
	<p>The X-bit is not affected by a division. The N-bit is set if the quotient is negative. The Z-bit is set if the quotient is zero. The Vbit is set if division overflow occurs (in which case the Z- and Nbits are undefined). The C-bit is always cleared.</p>

MULTF	Signed floating-point multiply
Operation:	$[\text{destination}] \leftarrow [\text{destination}] \times [\text{source}]$
Syntax:	MULTF <ea>,Dn
Description:	Multiply the destination operand by the source operand and store the result in the destination, interpreting operands and result as signed floating-point numbers.
Condition codes:	X N Z V C - * * * 0
	<p>The X-bit is not affected by a multiplication. The N-bit is set if the product is negative. The Z-bit is set if the product is zero. The V-bit is set if division overflow occurs (in which case the Z-bit and the N-bit are undefined). The C-bit is always cleared.</p>

AND	AND logical
Operation:	$[destination] \leftarrow [source].[destination]$
Syntax:	AND <ea>,Dn AND Dn,<ea>
Description:	AND the source operand to the destination operand and store the result in the destination location.
Condition codes:	X N Z V C - * * 0 0

The N-bit is set to the most significant bit of the result. The Z-bit is set if the result is equal to zero.

ASR	Arithmetic shift left/right
Operation:	$[\text{destination}] \leftarrow [\text{destination}] \text{ shifted by } \langle \text{count} \rangle$
Syntax:	ASL $\langle \text{ea} \rangle, \text{Dn}$ ASR $\langle \text{ea} \rangle, \text{Dn}$ ASL $\# \langle \text{data} \rangle, \text{Dy}$ ASR $\# \langle \text{data} \rangle, \text{Dy}$ ASL $\langle \text{ea} \rangle$ ASR $\langle \text{ea} \rangle$
Description:	<p>Arithmetically shift the bits of the operand in the specified direction (i.e., left or right). The shift count may be specified in one of three ways. The count may be a literal, the contents of a data register, or the value 1. An immediate (i.e., literal) count permits a shift of 1 to 8 places. If the count is in a register, the value is modulo 64 (i.e., 0 to 63). If no count is specified, one shift is made (i.e., ASL $\langle \text{ea} \rangle$ shifts the contents of the word at the effective address one place left).</p> <p>The effect of an arithmetic shift left is to shift a zero into the least-significant bit position and to shift the most-significant bit out into both the X- and the C-bits of the CCR. The overflow bit of the CCR is set if a sign change occurs during shifting (i.e., if the most-significant bit changes value during shifting). The effect of an arithmetic shift right is to shift the least-significant bit into both the X- and C-bits of the CCR. The most-significant bit (i.e., the sign bit) is replicated to preserve the sign of the number.</p>
Condition codes:	X N Z V C * * * * *

Bcc	Branch on condition cc
-----	------------------------

Operation: If $cc = 1$ THEN $[PC] \leftarrow [PC] + d$

Syntax: Bcc <label>

Description: If the specified logical condition is met, program execution continues at location $[PC] + \text{displacement}$, d.

BEQ	branch on equal	Z
BMI	branch on minus	N
BNE	branch on not equal	\bar{Z}
BRF	branch on overflow set	V

Condition codes: X N Z V C
- - - - -

BRA	Branch always
-----	---------------

Operation: $[PC] \leftarrow [PC] + d$

Syntax: BRA <label>
 BRA <literal>

Description: Program execution continues at location $[\text{PC}] + d$.

Condition codes: X N Z V C
- - - - -

LOAD	Load value
Operation:	[data register] \leftarrow <data>
Syntax:	LOAD <ea>,Dn LOAD #<data>,Dn
Description:	Write to data register, where the data depends on the addressing mode. With direct addressing, it is the memory contents at the given address. With immediate, the given literal.
Condition codes:	X N Z V C - - - - -
STORE	Store value
Operation:	[memory] \leftarrow <data>
Syntax:	STORE Dn,<ea> STORE #<data>,<ea>
Description:	Write to main memory. If a data register is given, its contents are written. If immediate addressing is used, a given literal is written.
Condition codes:	X N Z V C - - - - -
HALT	Halt execution
Operation:	HALT
Syntax:	HALT
Description:	Processor suspends all processing.
Condition codes:	X N Z V C 0 0 0 0 0