Grafritande Räknare - Designskiss

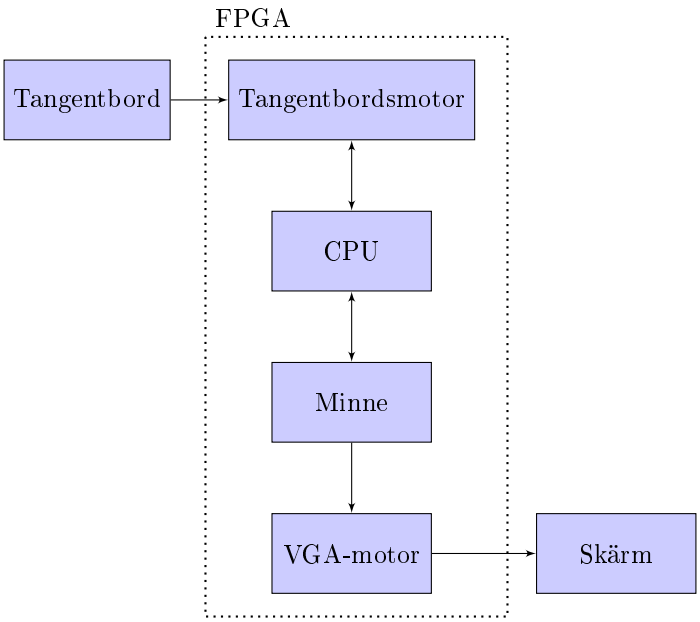
Hannes Haglund hanha265
Felix Härnström felha423
Silas Lenz sille914

23 mars 2016

Innehåll

1 Översikt

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).



2 CPU

Vår processor är mikroprogrammerad, med gemensamt 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 tangentbordläsning, och utritning av bildminnets innehåll. Dessa uppgifter inkluderar: beräkningar, historik, parsing av input, beräkning av graf, och så vidare.

3 Instruktioner

Vi har följande adresseringsmoder:

- Direkt (Binärkod `00`)
- Omedelbar (Binärkod `01`)
- Indirekt (Binärkod `10`)

Följande instruktionsmängd:

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

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

I programminnet ges en instruktion på följande format:

Beskrivning	Instruktion	Dataregister	Mod	Adress/literal
Storlek	5 bitar	3 bitar	2 bitar	22 bitar

Om vi nu exempelvis skulle vilja subtrahera 48672 från värdet i register 5 skulle vi skriva följande i minnet:

SUB	Register 5	Omedelbar	Värdet 48572
`01010`	`101`	`01`	`00000010111110110111100`

För mikrokodsinstruktionerna används följande format:

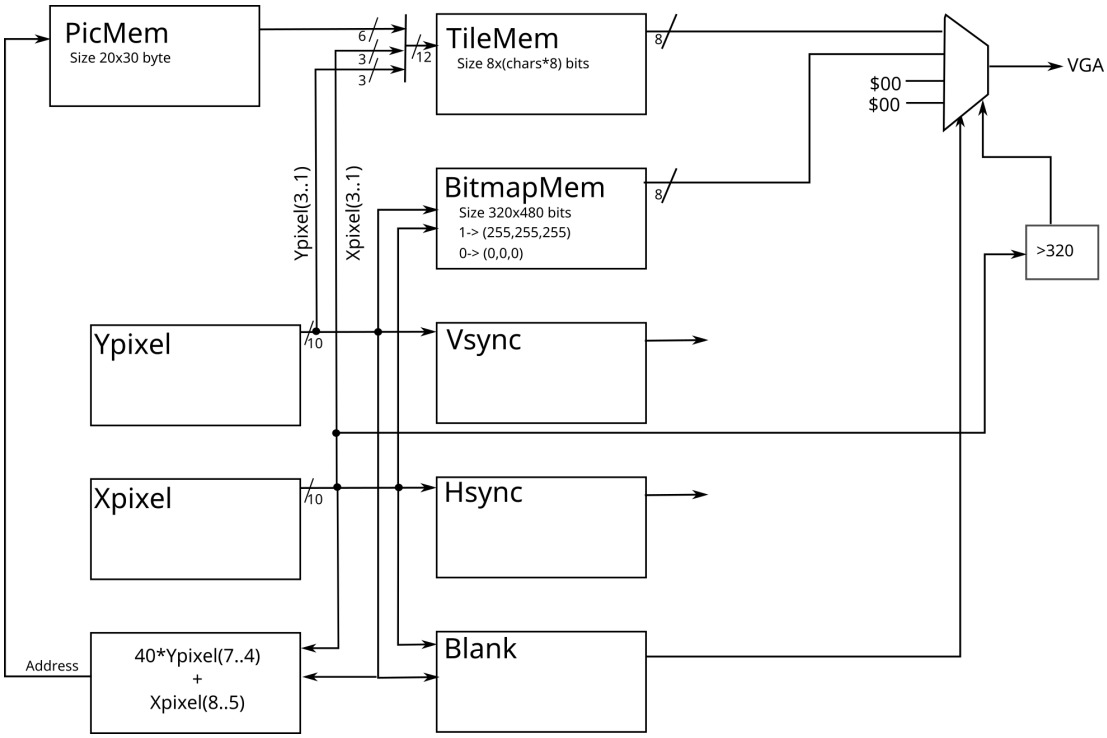
Beskrivning	ALU	Till buss	Från buss	PC	SEQ	μADR
Storlek	5 bitar	4 bitar	4 bitar	1 bit	4 bitar	14 bitar

4 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.



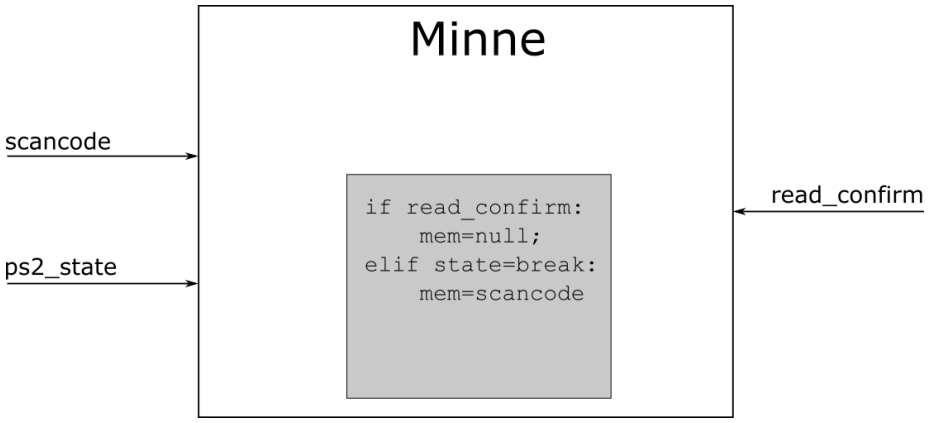
Figur 1: Blockschema över VGA-motor

5 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.

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



Figur 2: Blockschema över tangentbordsminne och read_confirm - logik

6 Minne

Vi har följande minnen:

- PC (rw)
- ASR (rw)
- IR (rw)
- μPC (rw)
- μMinne (rw)
- Programminne (rw)
- Dataminne (rw)
- 8 generella dataregister (rw)
- Statusregister (r)
- Bildminne
- Bitmapminne
- Tileminne

Där program och dataminne använder ordlängden 32 bitar.

7 Programmering

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

8 Milstolpe

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

Bilaga A Instruktionsuppsättning

HALT	Halt execution
Operation:	HALT
Syntax:	HALT
Binary code:	00000
Description:	Processor suspends all processing.
Condition codes:	X N Z V C 0 0 0 0 0

LOAD	Load value
Operation:	[data register] ← <data>
Syntax:	LOAD <ea>,Dn LOAD #<data>,Dn
Binary code:	00001
Description:	Write to data register, where the data depends on the adressing mode. With direct adressing, 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] ← <data>
Syntax:	STORE Dn,<ea> STORE #<data>,<ea>
Binary code:	00010
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 - - - - -

BRA — Branch always

Operation: `[PC] ← [PC] + d`

Syntax:
```
BRA <label>
BRA <literal>
```

Binary code: `00011`

Description: Program execution continues at location [PC] + d.

Condition codes:

X	N	Z	V	C
-	-	-	-	-

Bcc — Branch on condition cc

Operation: `If cc = 1 THEN [PC] ← [PC] + d`

Syntax: `Bcc <label>`

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

`BEQ` (00100)	branch on equal	`Z`
`BMI` (00101)	branch on minus	`N`
`BMI` (00110)	branch on not equal	$\overline{\texttt{Z}}$
`BRF` (00111)	branch on overflow set	`V`

Condition codes:

X	N	Z	V	C
-	-	-	-	-

ADD — Signed integer add

Operation: `[destination] ← [source] + [destination]`

Syntax:
```
ADD <ea>,Dn
ADD Dn,<ea>
ADD #<value>,Dn
```

Binary code: `01000`

Description: Add the source operand to the destination operand and store the result in the destination location. The source can be given as a literal using immediate addressing.

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 — Signed floating-point add

Operation: `[destination] ← [source] + [destination]`

Syntax:
```
ADDF <ea>,Dn
ADDF Dn,<ea>
ADDF #<value>,Dn
```

Binary code: `01001`

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. The source can be given as a literal using immediate addressing.

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 — Signed integer subtract

Operation: `[destination] ← [source] - [destination]`

Syntax:
```
SUB <ea>,Dn
SUB Dn,<ea>
SUB #<value>,Dn
```

Binary code: `01010`

Description: Subtract the destination operand from the source operand and store the result in the destination location. The source can be given as a literal using immediate addressing.

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 — Signed floating-point subtract

Operation: `[destination] ← [source] - [destination]`

Syntax:
```
SUBF <ea>,Dn
SUBF Dn,<ea>
SUBF #<value>,Dn
```

Binary code: `01011`

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. The source can be given as a literal using immediate addressing.

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: `[destination] ← [destination] / [source]`

Syntax:
```
DIVF <ea>,Dn
DIVF #<value>,Dn
```

Binary code: `01100`

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

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.

MULTF — Signed floating-point multiply

Operation: `[destination] ← [destination] × [source]`

Syntax:
```
MULTF <ea>,Dn
MULTF #<value>,Dn
```

Binary code: `01101`

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. The source can be given as a literal using immediate addressing.

Condition codes:

X	N	Z	V	C
-	*	*	*	0

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.

AND — Logical AND

Operation: `[destination] ← [source].[destination]`

Syntax:
```
AND <ea>,Dn
AND Dn,<ea>
AND #<value>,Dn
```

Binary code: `01110`

Description: AND the source operand to the destination operand and store the result in the destination location. The source can be given as a literal using immediate addressing.

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.

ASL/ASR Arithmetic shift left/right

Operation: `[destination] ← [destination] shifted by <count>`

Syntax:
```
ASL <ea>,Dn
ASR <ea>,Dn
ASL #<data>,Dy
ASR #<data>,Dy
ASL <ea>
ASR <ea>
```

Binary code: ASR: `01111`, ASL: `10000`

Description: 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 <ea> shifts the contents of the word at the effective address one place left).
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 leastsignificant bit into both the X- and C-bits of the CCR. The mostsignificant bit (i.e., the sign bit) is replicated to preserve the sign of the number.

Condition codes:

X	N	Z	V	C
*	*	*	*	*