Designspecifikation 4ss-troids

TSEA83, Grupp 34

Anton Wegeström - antwe841
Anton Östman - antos931
Jakob Österberg - jakos322
Rasmus Wallin – raswa151

Version 0.1

2022-02-24

Innehållsförteckning

Sammanfattning

I denna designspecifikation beskrivs en 2-stegs pipeline-processor med nödvändig kringutrustning. Tanken är att man ska kunna köra spelet asteroids på processorn där användaren kontrollerar ett rymdskepp via en joystick och spelplanen visas på en skärm via VGA. Senare i projektet ska UART inladdning av program läggas till, samt om tid finns så ska ljud implementeras.

Processorn

Processorn använder ett delat data- och program-minne. Dess huvudsakliga uppgift är att räkna ut vart sprites befinner sig och hantera I/O mellan spelaren och spelet. Registerfilen består av 16st register som vardera är 16 bitar stora, vilket också blir processorns naturliga ordbredd.

Instruktionsset

Mnemonic	**Funktion**
NOP	No operation
RJMP *offset*	Hopp till PC+1+*offset*
JSR *offset*	Subrutinanrop till PC+1+*offset*
RET	Återhopp till subrutin
BEQ *offset*	Hopp om Z = 1
BNE *offset*	Hopp om Z = 0
BPL *offset*	Hopp om N = 0
BMI *offset*	Hopp om N = 1
BGE *offset*	Hopp om N xor V = 0
BLT *offset*	Hopp om N xor V = 1
LDI *Rd,const*	RD <= konst
LD *Rd,Ra*	RD <= MEM(Ra)
ST *Rd,Ra*	MEM(Rd) <= Ra
COPY *Rd,Ra*	Rd <= Ra
ADD *Rd,(Ra/*	Rd <= Rd + Ra, uppd. Z,N,C,V
ADDI *Rd,const*	RD <= Rd + konst, uppd. Z,N,C,V
CMP *Rd,Ra*	Rd-Ra, uppd. Z,N,C,V
CMPI *Rd,const*	Rd-const, uppd. Z,N,C,V
AND Rd,Ra	Rd <= Rd AND Ra, uppd. Z,N,C,V
ANDI *Rd,const*	Rd <= Rd AND konst, uppd. Z,N
OR *Rd,Ra*	Rd <= Rd OR Ra, uppd. Z,N,C,V
ORI *Rd,const*	RD <= Rd OR konst, uppd. Z,N,C,V
PUSH *Ra*	DM(SP) <= Ra
POP Rd	Rd <= DM(SP+1)
Shifta/Rotera (vid behov)	
ADC *Rd,Ra*	Rd <= Rd + Ra + C
SBC *Rd,Ra*	Rd <= Rd – Ra – C
MUL *Rd,Ra*	Rd <= Rd * Ra (unsigned)
MULS *Rd,Ra*	Rd <= Rd * Ra (signed)

Eftersom vi vill ha plats med åtminstone 29st instruktioner, och vi kan tänkas behöva lägga till fler, används 6 bitar för att identifiera vilken instruktion som ska utföras. Vi kommer även ha tre adresseringsmoder och använder 2 bitar till detta. Dessa moder är omedelbar, registerbaserad och

direkt adressering. Instruktionsregistret delas upp enligt nedan beroende på typen av instruktion (se tabell ovan) och är 28 bitar brett.

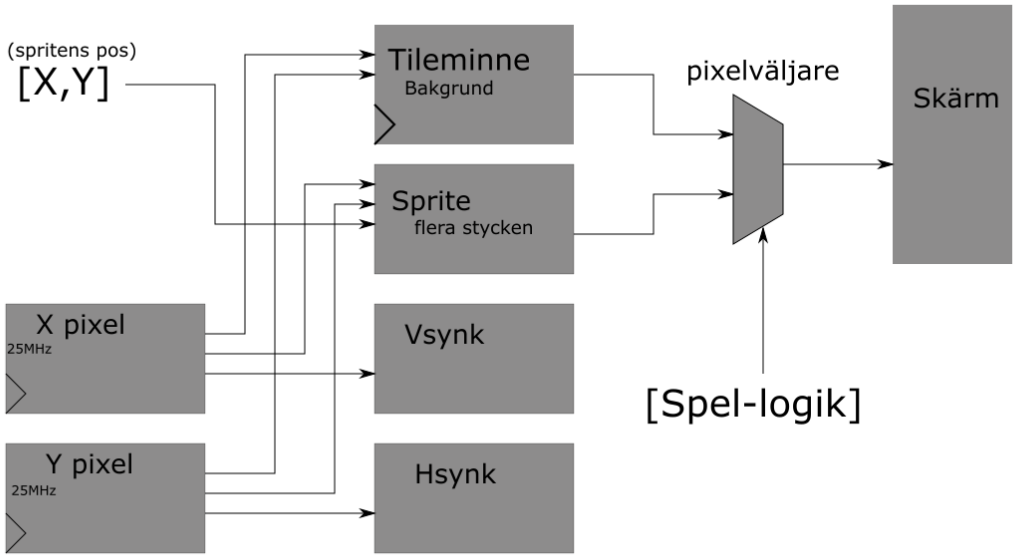
```
   Op.      M    Rd      Ra       Oanvänt
[000000][00][0000][0000][000000000000]

   Op.      M    Rd     Adress/Konstant
[000000][00][0000][0000000000000000]
```

Start av processor

Till en början har vi ett program laddat från start i programminnet. Vi vill senare i projektet använda oss av en boot-loader för att kunna ladda in program via UART.

Grafik

Vi har valt att ha upplösningen 640x480 med 4 bitar till färger. Vi har tänkt använda oss av sprites för spelobjekt och tiles för bakgrunden. Ett abstrakt blockschema ses i figur 1 nedan.



Figur 1: Blockschema över grafikdelen.

I/O

Vi kommer använda en Joystick (Pmod JSTK) för att bestämma spelarens position. Kommunikation sker över SPI och vi måste koppla FPGA:n mot joystickens MOSI/MISA/SCK/CS pins. UART kommer användas för att ladda över program till FPGA:n i ett senare skede, detaljer för kopplingen till FPGA:n tar vi från labb 3. Detaljer för VGA tas från labb 4 (samt se grafikdelen ovan). En stereohögtalare kommer användas för att spela ljud relaterat till spelet, implementeras om vi hinner med övriga delar av projektet. Kommunikationen sker då över I2S protokollet på pins kopplade mellan högtalaren och FPGA:n. I/O-enheterna kopplas till de högsta CPU registren.

Minnesanvändning

Till vår konstruktion vill vi ha ett bildminne, ett tileminne, fem olika sprites, programminne och dataminne. Tileminnet adresseras med 13 bitar vilket ger plats för 32st 8x8px stora tiles. Bredden på tileminnet är 4 bitar som representerar varje pixels färg. Det tillhörande bildminnet adresseras också med 13 bitar, där vi använder 4800 adresser för att peka ut tiles i tileminnet, vilket gör att minnet har bredden 13 bitar.

Våra fem olika sprites ska representera tre olika storlekar av asteroider, spelaren och skott från spelaren. Asteroiderna är dels den största som är 64*64px och adresseras med 10 bitar, dels den mellersta som är 32x32px och adresseras med 8 bitar och dels en minsta som är 16*16px och adresseras med 6 bitar. Spelaren är 32x32px och adresseras med 8 bitar och ett skott från spelaren är 8*8px och adresseras med 4 bitar.

Program- och data-minnet adresseras båda med 16 bitar men programminnet är 28 bitar brett (likt en instruktion) medan dataminnet är 16 bitar. Därmed sätter vi alltså PC, SP och ADR till 16 bitar. Vi måste också ha ett statusregister som behöver rymma fem flaggor, men vi gör den 3 bitar stor. Och om vi mot förmodan får slut på programminne kan vi använda två register för att adressera ett 32 bitars minne. De minnen som CPU:n skriver eller läser ifrån är: PC, SP, ADR, dataminne, programminne.

Programmering

Vi programmerar för hand till en början, det vill säga i maskinkod. När projektet börjar ta form så skapas en assembler som gör om mnenomics till maskinkod.

Timing

Vi klockar vsync pulser och utför spelloopen efter ett visst antal sådana.

Halvtidsmål

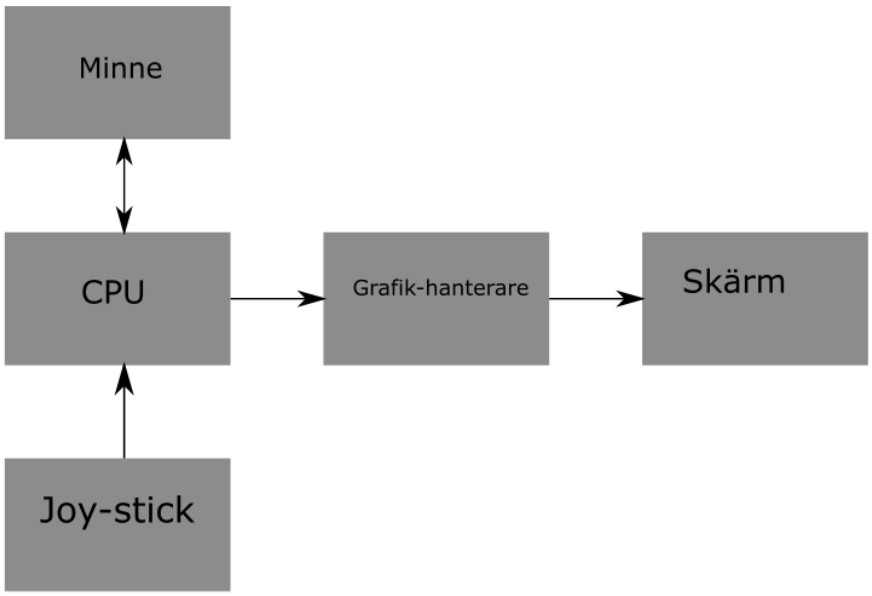
Vi vill kunna rita ut någonting på en skärm och i simulering visa upp en fungerande processor.

Blockschema för designen

Nedan visas ett blockschema över CPU:n och dess kringliggande hårdvara.

Ingående delar:

- ALU
- Register
- PC
- Minne (Tile/Sprite/Data/Program)
- IR1, IR2
- Hopp logik, HOP-NOP
- Data forwarding?
- Flags (SR)
- IR
-



Figur 2: Blockschema över CPU:n och kringliggande hårdvara.