Designspecifikation Epik Snake

TSEA83, G48

version: 0.1

Edvard Wetind	edvwe024@student.liu.se
Hugo Nilsson	hugni385@student.liu.se
Christoffer Näs	chrna581@student.liu.se
Mikael Lundgren	miklu523@student.liu.se

Sammanfattning

Implementation

Programminne
Två delar. En “ram del” som uppdateras under spelets gång, typ ormens längd.
En “instruktions del” som sparar instruktionerna och programmeringen för spelet.

Processor
Själva logiken i programmet. Lägger till frukter, förlänger ormen, tar hand om varje tick i spelet. Varje tick styrs av interrupt.

Grafikmotor
Skriver till skärmen vad som ska visas och var. Uppdateras varje tick.

Bildminne
Sparar “bilder” på hur de olika “pixlarna” ska se ut. T.ex. vilken färg och hur ormen ska se ut

Snake Controller
Tar hand om input och output från kontrollen. Avkodar och skickar till processorn vad spelaren vill att ormen ska göra.

CPU

Vi har tänkt att bygga en mikroprogrammering processor. Processorn kommer använda sig av delat programminne, en del för instruktioner och en RAM del som innehåller ormens position och var frukter finns.

Instruktions set

Mnemonic	Funktion
NOP	No operation
RJMP	offset Hopp till PC+1+offset
JSR	offset Subr.anrop till PC+1+offset
RET	˚Aterhopp fr˚an subrutin
IRET	˚Aterhopp fr˚an avbrott
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 ⊕ V = 0
BLT offset	Hopp om N ⊕ V = 1
LDI Rd,const	Rd<=const
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+const, uppd. Z,N,C,V
CMP Rd,Ra	Rd-Ra : uppdatera Z,N,C,V
CMPI Rd,const	Rd-const : uppdatera Z,N,C,V
AND Rd,Ra	Rd<=Rd AND Ra, uppd. Z,N,C,V
ANDI Rd,const	Rd<=Rd AND const, uppd. Z,N,C,V
OR Rd,Ra	Rd<=Rd OR Ra, uppd. Z,N,C,V
ORI Rd,const	Rd<=Rd OR const, uppd. Z,N,C,V
PUSH Ra	DM(SP)<=Ra
POP Rd	Rd<=DM(SP+1)

Processorn kommer att ha en instruktionsbredd på 32 bitar. Vi kommer enbart att använda oss av direkt adressering. Våran processor kommer att laddas direkt från start. CPUn kommer att hålla koll på ormen, samt stadiet i spelet. Uppdatera spelstadiet varje tick.

Grafik

Vi kommer att använda oss av väldigt simpel grafik. Ormen och äpplet är svart, resten vitt. Ormens positioner samt äpplets är sparat i RAM.
VGA_motor ritar svart eller vit beroende på om ormen är på den pixel VGA_motorn ska rita ut. Våran upplösning kommer att vara 400x400, då varje tile är 10x10 pixel. Det vill säga en spelplan på 40x40 tiles. Äpplet är 1 tile stor. Ormen är längd x tile stor. med svart och vitt. Grafik Enheten kommer att samarbeta med CPU-n genom att VGA_motorn frågar RAM ifall ormen/äpple är på current pixel. om det ritar den orm/äpple, annars bakgrund.

I/O-enheter

Vår I/O-enhet är en joystick med SPI för att låta användaren styra ormen. Vår tilldelade NEXYS-3 innehåller all hårdvara ur blockschemat förutom joystick:en. Joysticken kommer att kopplas in med hjälp av SPI. Den kopplas in i en minnescell som finns på NEXYS-3.

Minnesanvändning

Vi har ett RAM minne som innehåller bland annat ormens position samt ett program minne som innehåller alla instruktioner. Ett PM (Program memory) som innehåller alla instruktioner. Ett IR minne (intstruktions register), innehåller den laddade instruktioner ifrån PM.

Våra olika minnen kommer att behöva ha storleken:

- Joystick: 4-bitars register.
- SR, statusregistret kommer att behöva rymma 5 flaggor men vi gör den till 8 bitar för enkelhetens skull.
- Instruktionsregistret och program minnet kommer att behöva vara 28 bitar brett.
- Programminnet kommer att behöva 4-bitar för I/0.
  40*40(1(huvud)+1(orm)+1(äpple)+2(position av förra ormsegmentet) + 3(övrigt)) = 12 800 bitar = 1,6 kB för grafik och status.

I första hand kommer CPU-n försöka läsa från cachen. Om inte informationen finns i cache, kommer processorn istället att läsa ifrån primärminnet och därefter kopiera denna information till cachen för snabbare access nästa gång. Minnet för FPGAn kommer att vara tillräckligt stort för det vi tänker att implementera.

Programmering

För programmeringen av spelet så kommer en (game)loop behöva programmeras i PM. Där finns olika “sektioner” som körs beroende på vad som händer i spelet/vilka flaggor som är satta. Det vill säga, assemblerkod med subrutiner. Vi kommer att hårdkoda i mikroprogrammering i datorn men använda oss av assembler instruktioner för att ta fram mikrokoden.

Timing

Processorn klockas i 100 Mhz en uppdateringsfrekvens på 2fps uppnås genom att ta processor klockan på 100 Mhz modulo 50 M. Kör nästa “tick”s instruktioner på en gång efter en uppdatering till skärmen. Men väntar tills skärmen har uppdaterats innan den kör nästa “tick”s instruktioner. Vi kommer att ha en vänte loop i programmet, för att vänta på bilduppdatering. Alltså kommer programmet att behöva vänta in nästa tick.

Milstolpar

Som halvtids mål har vi en färdigbyggd dator som klarar av någonting grafiskt och I/O relaterat.

Blockscheman