DESIGNSPEC

åL

TSEA83

Elliot Norlander, ellno907
Jennifer Santos, jensa682
Jacob Sjölin, jacsj573
Elin Rydebrink, eliry213

Version 4, 23/5-2023

Sammanfattning

Denna designspecifikation är menad att ge en övergripande bild av konstruktionen av en mikroprogrammerad processor. Utöver processorn innehåller också konstruktionen en avkodningsmodul för inläsning av en joystick samt en VGA-motor för generering av VGA-baserad bilddata.

Implementation

I detta projekt kommer det användas flera olika typer av hårdvara. Till exempel kommer FPGA-kortet vara ansvarig för VGA-motorn, åL-controller, programminnet och bildminnet. Joysticken kommer göra det möjligt att styra ålen. Det kommer även användas en VGA-monitor för att visualisera spelet.

CPU

Processorn ska kunna hantera 32-bitars instruktioner. Det ska finnas plats för instruktionsnotering (AND, ADDI, etc.), två register samt flaggor. Med de ovannämnda specifikationerna är 16-bitars instruktioner troligtvis inte tillräckligt och därav har beslutet fattats att 32-bitars instruktioner kommer användas i denna processor. Gällande stödda adresseringsmoder skall det finnas stöd för direkt, indirekt samt registerbaserad adressering. Processorn kommer inneha ett förladdat program, då ingen UART modul kommer implementeras.

Grafik

Planen är att programmet kommer använda bildminnesdriven grafik. För att få in färg, används en bitmap med två bitar. På så sätt kommer fyra olika färger kunna användas. Värdet för varje pixel kommer att sparas bitvis i minnet. Vid visning på VGA-skärmen skall en upplösning på 640 x 480 pixlar användas, se illustration 1. Detta kommer underlätta implementationen, då en förkonstruerad VGA-modul finns från tidigare utförda laborationer i kursen.

Spelplanen kommer vara uppdelad i 20 x 15 spelrutor. För att generera bilddatan kommer tile-baserad grafik användas. Tanken är att använda ett 1:1 samband mellan en grafisk tile och en ruta i spelet. Med andra ord kommer varje ruta i den rutiga bakgrunden vara en tile och en “enhet” av ålen kommer vara en tile. Tanken är att detta ska underlätta eventuell synkronisering mellan spelmotorn och grafiken.

Då tile-baserad grafik används behöver VGA-motorn inte synkroniseras med CPUn. Tanken är att VGA-motorn löpande genererar bilddatan, och att CPUn styr när bilddatan faktiskt ska skickas från bildminnet till VGA-monitorn. Detta sker med en preliminär frekvens på 60 Hz. Då CPUns klocka är hög i jämförelse med uppdateringsfrekvensen för skärmen kommer en separat signal användas för att diktera när VGA-datan ska skickas.

I/O-enheter

För att tolka input från joysticken kommer det behövas en avkodningsmodul som kan läsa av joystickens signaler. Den avkodade datan kommer sedan att lagras i ett avsatt register.

Outputen till VGA-monitorn kommer att hanteras av VGA-motorn i kombination med CPUn. Inputen från joysticken kommer att lagras i CPUn och bilddatan kommer att sparas i grafikminnet.

Minnesanvändning

Konstruktionen består av ett bildminne, ett instruktionsregister (se figur 3), ett flaggregister samt ett programminne där programmet och information om spelet sparas, till exempel ormens koordinater eller fiskens koordinater. Vidare kommer även ett timing-register finnas. Funktionaliteten kring detta förtydligas under rubriken “Timing”. För att estimera minnesanvändningen har följande uppskattning gjorts:

Bildminne - 225 rader X 32 bitar
Instruktionsregister - 364 rader X 8 bitar
Flaggregister - 4 rader X 1 bit
Programminne - 1024 rader X 16 bitar
Timing-register - 1 rad X 32 bitar

Eftersom CPUn läser och skriver i bildminnet kommer Block-RAM att användas för detta. Om den tekniska dokumentationen för FPGA-kortet uppfattats rätt, finns ett Block-RAM om 576 Kbits = 72 Kbyte att tillgå. Ett bildminne på 225 X 32 = 7200 bits = 900 byte = 0.9 Kbyte kommer behövas. Eftersom 0,9 Kbyte < 72 Kbyte kommer detta enstaka Block-RAM vara tillräckligt för att spara informationen i bildminnet.

Programmering

Planen är att skapa ett set av instruktioner med hjälp av mikroprogrammering. Instruktionerna skall sedan användas i syfte att skapa de individuella delarna av åL.

åL-controller är spelets huvuddel. Kollisionshantering kontrolleras här och annan funktionalitet som möjliggör ett fullt fungerande spel i enlighet med spelets regler.

Bildminnet sparas i block-RAM eftersom CPU-n måste kunna skriva och läsa i detta minne. Inuti bildminnet sparas ett tile-nummer som kopplas till varje enskild bildruta i spelplanen.

I VGA-motorn sparas tileminnet med information om hur varje tile ska se ut. VGA-motorn genererar bild- och synksignaler, vilket resulterar i att spelplanen ritas upp.

Timing

För att spelet ska ha en rimlig hastighet är tanken att implementera en vänteloop som kontinuerligt läses av från fördröjningsregistret. Genom att t.ex. ladda in talet 25M i ett register, och sedan subtrahera ett från detta tal tills vi når noll, kan man med en processorklocka på 100 MHz och en subtraktionsoperation som tar 1 cykel, fördröja spelflödet med t.ex. 0,25 sekunder. Denna fördröjning kan konfigureras med vilket 32 bitars tal som helst, det behöver alltså inte vara just 25M. Tanken är att denna funktionalitet implementeras som en separat timermodul.

Instruktionsuppsättning

Ett set av mikroprogrammerade instruktioner kommer skapas. Se figur 1.

Milstolpe

Efter halva projektperioden ska en fungerande simulering av CPU:n med vissa implementerade instruktioner kunna visas. Vi hoppas också kunna visa en tile-baserad bild på monitorn.

Blockschema

Se figur 2.

Appendix

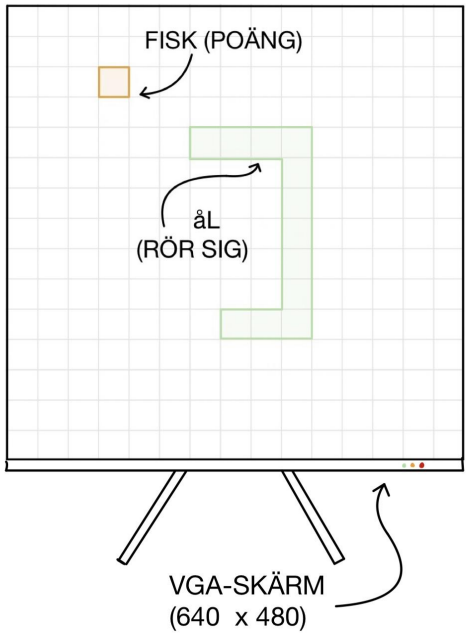
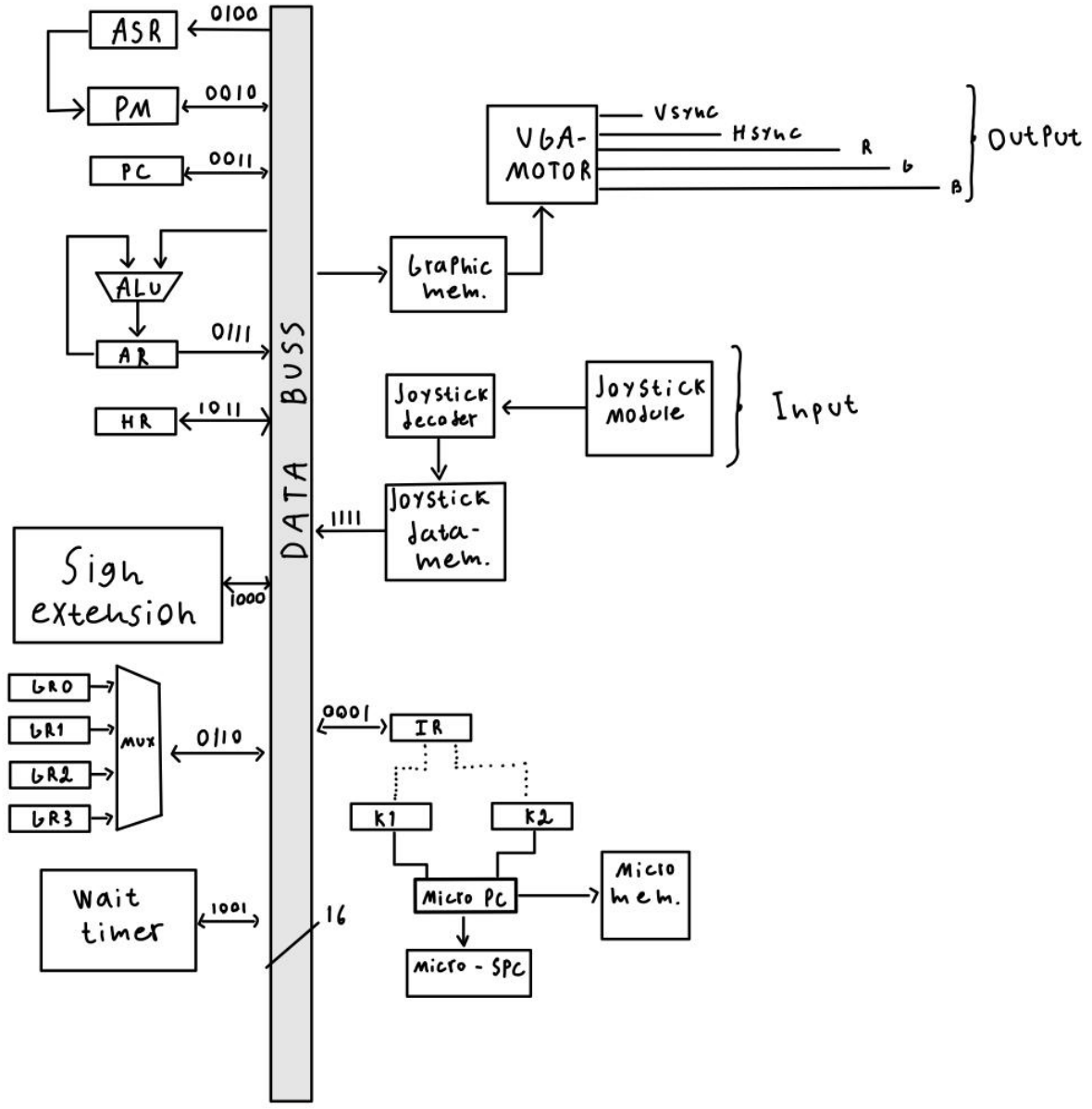


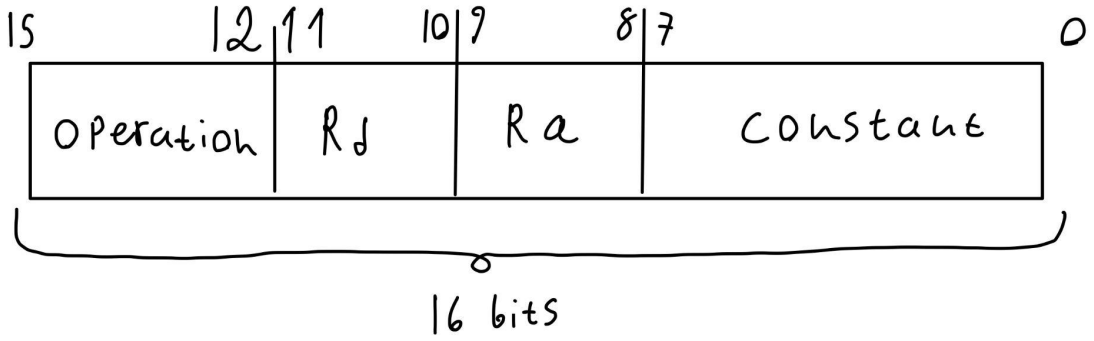
Illustration 1, översiktsbild av spelupplevelsen

Mnemonic	**Funktion**
ADD. Rd, Ra	Rd <= Rd+Ra, uppdatera flag
ADDI. Rd, const	Rd <= Rd+const, uppdatera flag
AND. Rd, Ra	Rd <= Rd AND Ra, uppdatera flag
ANDI. Rd, const	Rd <= Rd AND const, uppdatera flag
BEQ. offset	Hopp om Z = 1
BNE. offset	Hopp om Z = 0
BMI. offset	Hopp om N = 1
BGE. offset	Hopp om N = 0
CMP. Rd, Ra	Rd - Ra uppd. flag
CMPI. Rd, const	Rd-const (Z,N,C,V)
COPY. Rd, Ra	Rd <= Ra
JSR. offset	Subr.anrop till PC+1+offset
LD. Rd, Ra	Rd <= MEM(Ra)
LDI. Rd, const	Rd <= const
NOP	No operation
OR. Rd, Ra	Rd <= Rd OR Ra (Z, N, C, V)
ORI. Rd, const	Rd <= Rd OR const (Z, N, C, V)
RET	Återhopp från subrutin
BRA	Hopp till PC+1+offset
ST	MEM(Rd) <= Ra

Figur 1, tillåtna instruktioner



Figur 2, blockschema av processorn



Figur 3, instruktionsregister