# Designspec - Svarte Jakob

Designspecifikation Svarte Jakob
TSEA83, Grupp 39
*Alfred Sjöqvist - alfsj019*
*Sebastian Karlsson - sebka786*
*Oliver Brenner Hedmark - olibr806*
*Isak Song Olsson - isaso926*

Version: 0.1

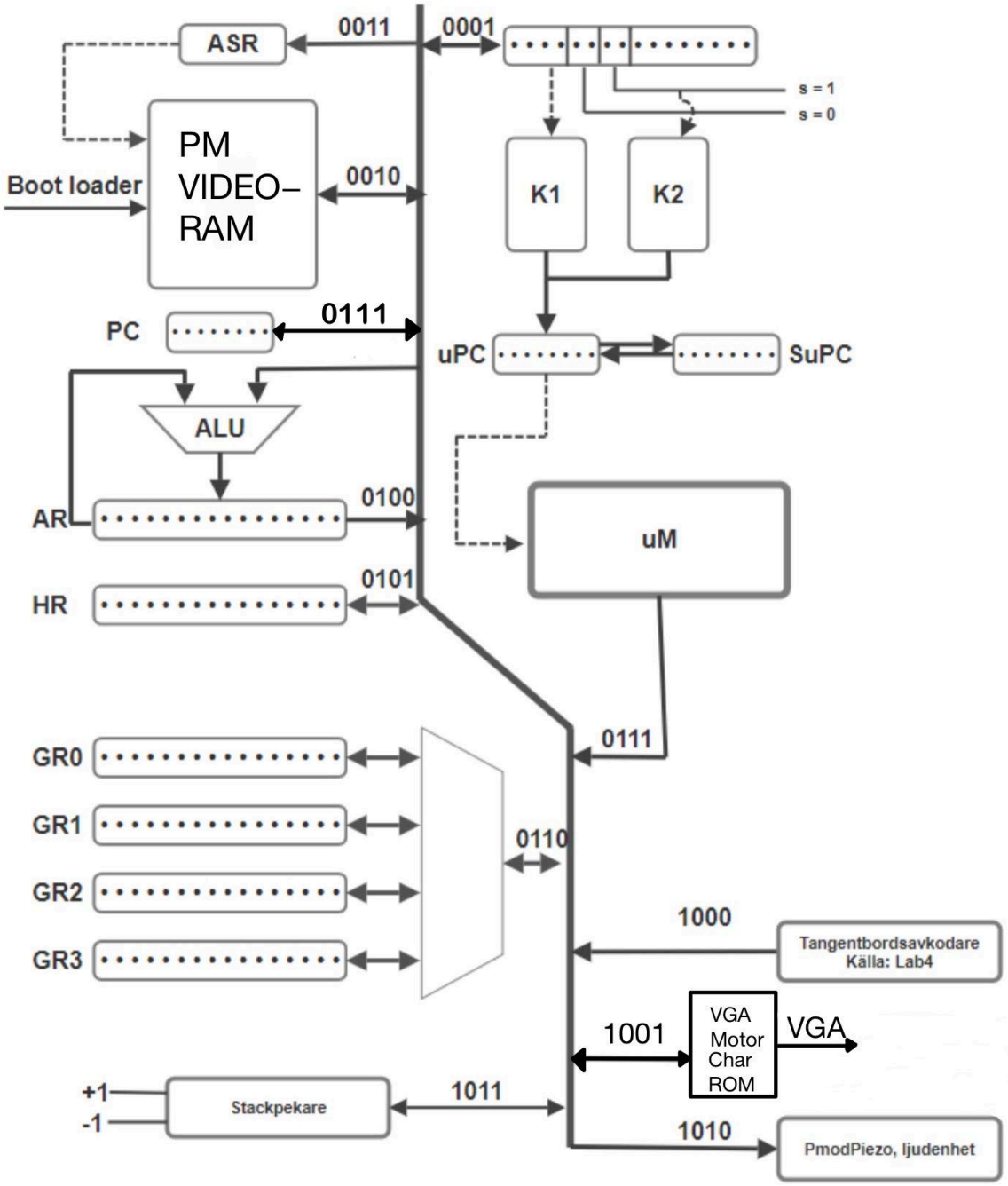
Datum: Version 0.1 [2024-02-26] Version 1.0 [2024-03-04]

Examinator: Anders Nilsson

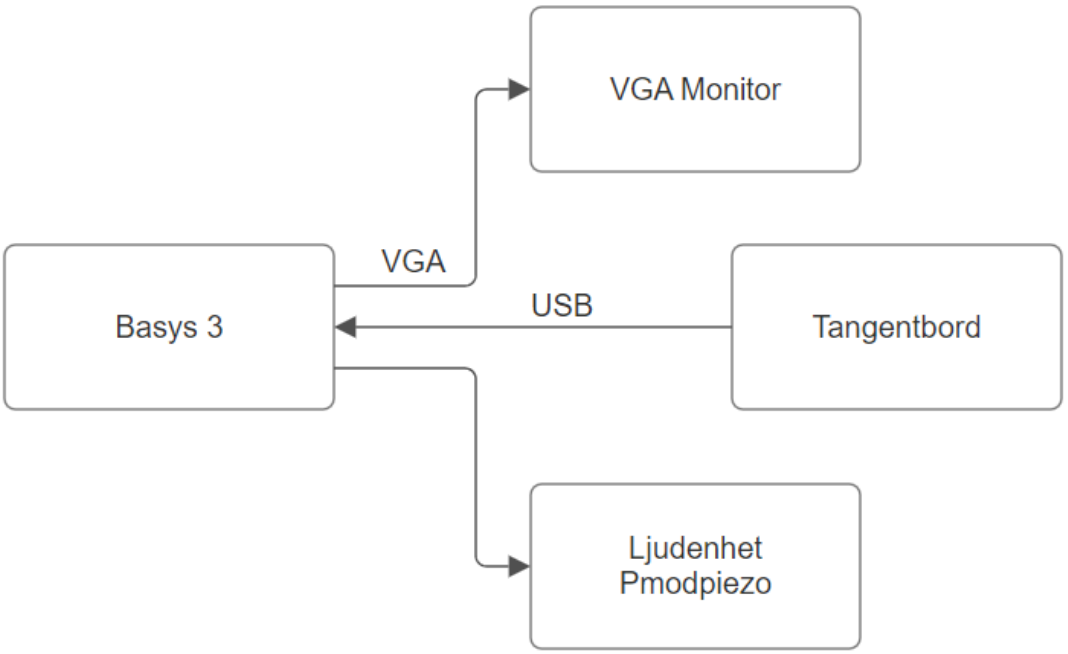
## Sammanfattning

Detta dokument beskriver designen av en mikroprogrammerad dator anpassad för spelet Svarte Jakob (Blackjack). Målet är att skapa en flexibel och effektiv lösning som kan hantera spelets logik och interaktion med användaren genom ett textbaserat gränssnitt, med potentiella utvidgningar till ett grafiskt gränssnitt och flerspelarstöd.

# 1. Blockschema




Figur 1. Visar blockschema.




Figur 2: Visar hårdvarukonfiguration.

# 2. Minnesanvändning.

Minnena i vår design är de som ses ovan i blockschemat. Det minnet som behövs till vår implementation bör finnas i FPGA:n. Om minnesproblem skulle uppstå skulle vi kunna reducera antalet befintliga kortlekar. Mikrominnet och K-registrerna behöver bara läsas från (dvs ROM) och byggs därmed som tabeller i VHDL medan resterande minnen och register tar vara på kortets Blockram.

# 3. Implementation

### 3.1 Hårdvarukomponenter

Till detta projekt kommer hårdvarukomponenterna bestå av följande:
En VGA skärm vars roll är att grafiskt visa spelet.
Ett tangentbord vars roll är att hantera inputs för spelets logik.
Ett Basys-3 kort som fungerar som datorn för att köra spelet.
Pmod_I2S högtalare för att spela upp ljud under spelet.
Hårdvaran kopplas sedan in via buss.

### 3.2 Specifika Utmaningar

En utmaning spellogiksmässigt är hur vi hanterar korten. 6st kortlekar kommer användas med 52 kort vardera. Dessa 312 kort lagras med modellen delat minne i avsatt plats på primärminnet. Korten blandas senare med hjälp av en shufflevektor som körs godtyckligt många gånger för olika resultat.

Korten själva representeras av 7 bitar vardera eftersom detta underlättar skapandet av tile-grafiken för både kort och text.

# 4. CPU

### 4.1 Processormodell.

Mikroprogrammerad utökning av Björn Lindskog-datorn. Här har vi kopplat in samtliga input samt output-enheter via bussen.

### 4.2 Registerbredd och minnesbredd

Register- samt minnesbredd är satt till 16 bitar.

### 4.3 Programräknare

Programräknaren är likt Björn Lindskogs-datorn 16 bitar, vilket gott borde räcka till att implementera spelet på.

### 4.4 Instruktionsregister

Instruktionsregistret behöver vara kapabelt att rymma alla nödvändiga kombinationer av register och konstanter som krävs för varje instruktion. Instruktionsavkodaren fungerar som ett kombinatoriskt nätverk som, baserat på den specifika instruktionen, genererar styrkommandon till de olika delarna av processorn. Dessa kommandon aktiverar sedan de relevanta delarna för att utföra den önskade funktionen av instruktionen. Vilka specifika styrkommandon som behövs beror på vilka instruktioner som är implementerade.

### 4.5 Stackpekare

Programmet kan ställa in stackpekaren, vanligtvis till adressen vid slutet av dataminnet. Efter detta hanterar den sig själv. Det innebär att den automatiskt minskar med ett vid anrop till subrutiner och vid avbrott, och ökar med ett när den återgår till tidigare position. Instruktioner som placerar eller hämtar värden från stacken orsakar samma automatiska justering av stackpekaren.’

### 4.6 Processorstart

Med hjälp av en bootloader laddas primärminnet genom UART för att starta processorn. Implementationen av detta kommer att vara enkel då FPGA:ns blockram är av tvåportstyp vilket ger oss flexibilitet i datahanteringen genom att möjliggöra samtidig läsning och skrivning vilket minskar initieringstiden.

### 4.7 Adresseringsmoder

Vi tänkte använda oss av CISC och kommer använda oss av samtliga fyra adresseringsmoder för M-delen av instruktionsregistret.

### 4.8 Instruktionsuppsättning

**NOP =** Ingen operation

**IRET** = Återhopp från avbrott

**RET** = Återhopp från subrutin

**RJMP =** Offset-hopp till PC+1+offset

**JSR** = Offset-subrutinanrop till PC+1+offset

**BNE** = Offset-hopp om Z = 0

**BPL** = Offset-hopp om N = 0

**BGE** = Offset-hopp om N $\oplus$ V = 0

**BEQ** = Offset-hopp om Z = 1

**BMI** = Offset-hopp om N = 1

**BLT** = Offset-hopp om N $\oplus$ V = 1

**ST** = Rd, Ra | MEM(Rd)<=Ra

**LDI** = Rd, const | Rd<=const

**LD** = Rd, Ra | Rd<=MEM(Ra)

**COPY** = Rd, Ra | Rd<=Ra

**ADD** = Rd, Ra | Rd<=Rd+Ra, uppdaterar Z, N, C, V

**ADDI** = Rd, const | Rd<=Rd+const, uppdaterar Z, N, C, V

**CMP** = Rd, Ra | Rd-Ra : uppdaterar Z, N, C, V

**CMPI** = Rd, const | Rd-const : uppdaterar Z, N, C, V

**PUSH** = Ra | DM(SP)<=Ra

**POP** = Rd | Rd<=DM(SP+1)

**AND** = Rd, Ra | Rd<=Rd AND Ra, uppdaterar Z, N, C, V

**ANDI** = Rd, const | Rd<=Rd AND const, uppdaterar Z, N, C, V

**OR** = Rd, Ra | Rd<=Rd OR Ra, uppdaterar Z, N, C, V

**ORI** = Rd, const | Rd<=Rd OR const, uppdaterar Z, N, C, V

# 5. Grafik

Som milstolpe skapar vi ett textbaserat gränssnitt för att kunna testa spelmekaniken till en början. Vidareutvecklingen är sedan att utveckla grafisk utritning för spelet. Tanken är då att införa tilegrafik på en grön bakgrund. Fördelen är att vi då lättare kan representera korten såväl som eventuell text på ett statiskt vis. Kortens tile-grafik lagras på char-rom men deras information lagras i primärminnet. Om vi vill införa animationer vid ett senare skede som exempelvis rörelser och vändning av kort så är tanken att lägga till så att en del av grafiken är spritebaserad.

# 6. I/O-enheter

Till input använder vi oss av ett tangentbord anslutet via USB. Avkodningen sker då med hjälp av assembly-instruktion för input. Detta kan ske genom att ha en flagga för tangenttryck.

Till output används en VGA-skärm till ett textbaserat spel som i mån av tid får ett grafiskt gränssnitt. Även ljudenheten PmodPiezo ansluts till Basys-kortet.

# 7. Minnesanvändning

För att hantera en VGA-upplösning på 640x480 pixlar krävs det att datorn kan adressera positioner i X- och Y-axeln var som helst i bilden, därför väljs registerbredden till 16 bitar. Detta antas underlätta hantering av tal i allmänhet. Register som R0, R1, ..., stackpekaren (SP), programräknaren (PC) och sprite-register (det vill säga register som kan adressera något) kommer då att ha en bredd på 16 bitar. På samma sätt blir dataminnet lämpligen också 16 bitar brett.

# 8. Programmering

Programmeras med hjälp av maskinkod. Assembler av något slag kan komma att implementeras om tid finns.

# 9. Timing

Spelet kommer synkroniseras enligt processorns klocka. Spelet i sig kommer inte behöva interrupts eller ett tick-baserat klocksystem då alla händelser sker sekventiellt från inputs från tangentbordet. Likaså med animationer och tid för deras utritning.

# 10. Milstolpe

Att ha en fungerande spellogik utan utritning och ljudeffekter. För detta krävs även en fungerande datorarkitektur.