Designspecifikation Ballongrally
TSEA83, Grupp 99

Anna Andersson - annan123
Bengt Bengtsson - benbe456
Carl Carlsson - carca789
Version 0.2

February 13, 2018

Sammanfattning

Denna designspec visar konstruktionen av en generell processor, uppbyggd via en 2-stegs pipeline, se figur 2. Processorn kan liknas vid en mikrokontroller, och är en RISC-konstruktion med delat program- och data-minne (Harvard-konstruktion) och har hårdvarustöd för subrutiner och avbrott. Utöver själva processorn innehåller även konstruktionen en tangentbordsavkodare för PS/2, speciella sprite-register samt en VGA-motor med tillhörande bildminne. Samtliga dessa delar är dock addressavkodade för att inte behöva speciella I/O-instruktioner.

Samtliga instruktioner i denna processor kommer att gå på en klockcykel, undantaget hopp som kommer att ta två klockcykler.

Processorn innehåller egentligen bara tre adresseringsmoder. Omedelbar adressering där data finns i instruktionsordet, registerbaserad adressering där data finns i ett register och direkt adressering där ett register innehåller en adress som pekar ut var data finns.

Processorn

Registerbredd och minnesbredd

Med utgångspunkt från att datorn ska klara en VGA-upplösning på 640x480 pixlar och att det helst ska vara så generellt att det går att adressera en position i bilden i X- och Y-led var som helst så behöver processorn kunna hanter tal om minst 10 bitar. Den generella registerbredden väljs därför till närmaste större två-potens, dvs 16 bitar, då det troligen underlättar hanteringen av tal i allmänhet. Dvs, register såsom R0, R1, ... (i registerfilen), SP, PC och sprite-register (dvs register som kan adressera något) blir då av 16 bitars bredd.

Data-minnets bredd blir lämpligen då också 16 bitar.

Eftersom instruktionerna ska utföras på en klockcykel så måste all information för en instruktion finnas tillgänglig i ett och samma instruktionsord. Dvs instruktionsregistret IR måste vara ganska brett. Vi räknar med att det går åt 28 bitar till IR för att hantera de kombinationer av instruktioner som vi vill kunna göra. Se avsnittet om IR för mer information. Programminnet blir av samma bredd som IR, dvs 28 bitar.

I/O-registret `KBD_DATA` blir 8 bitar brett, eftersom data från tangentbordet kommer byte-vis.

Statusregistret SR behöver rymma fem flaggor, men kan också få bli 8 bitar brett.

Programräknaren(PC) och programminnet

Programräknaren kan räknas upp med +1 (för att komma till nästa programrad), laddas med en hårdkodad adress (1, 2, 3, ...) vid avbrott, eller med ett värde från bussen (vid hopp eller subrutin). Dvs, programminnet börjar med ett antal s k vektorer, där adress 0 är reset-vektorn som innehåller en hoppinstruktion till själva huvudprogrammet, adress 1 är avbrottsvektorn för ett nytt tangentvärde och ska alltså innehålla en hoppinstruktion till den avbrottsrutinen. På så sätt blir det lätt att lägga till fler avbrott och avbrottsvektorer.

Instruktionregister(IR) och Instruktionsavkodare

Instruktionsregistret måste kunna innehålla nödvändiga kombinationer av register och konstanter för alla instruktioner. Instruktionsformatet fungerar enligt figur 1.

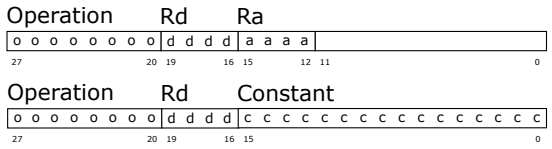

Figure 1: Instruktionsformat

Instruktionsavkodaren är bara ett kombinatorisk nät som beroende på instruktion kommer att ge styrsignaler till processorns olika delar och aktivera dessa för att uppnå önskad instruktionsfunktion. Exakt vilka styrsignaler som behövs blir en följd av vilka instruktioner som implementeras.

Dataminnet

Dataminnet används som arbetsminne och temporärlagring av register vid både subrutiner och avbrott. Dataminnet adresseras av något generellt register (R0, R1, ...) via vanliga instruktioner, eller stackpekaren SP vid subrutiner eller avbrott, och därför finns en särskild dataväg från PC in till dataminnet för att kunna lagra återhoppsadressen.

Registerfil, ALU och statusregister(SR)

Registerfilen innehåller processorns samtliga generella register (R0, R1, ...). Det exakta antalet är inte bestämt, men blir antagligen runt 16 register. Två register kan läsas, eller ett läsas och ett skrivas samtidigt. Dvs, instruktioner kan använda sig av ett destinationsregister och ett källregister samtidigt. Resultatet av instruktionsoperationen kan via ALU:n skrivas direkt till destinationsregistret. ALU:n kommer också att uppdatera statusregistrets statusflaggor (Z, N, C, V) för de instruktioner som påverkar dessa. Statusregistret innehåller också en spärrflagga (I) för avbrott. Spärrflaggan sätts automatisk då ett avbrott sker, för att förhindra ytterligare avbrott, och återställs automatiskt när avbrottsrutinen avslutar. Det är i nuläget inte möjligt att påverka innehållet i statusregistret via programmet, men den funktionen kanske läggs till senare vid behov.

Stackpekare (SP)

Stackpekaren kan sättas via programmet, då typiskt till adressen vid slutet av data-minnet. Därefter sköter den sig själv. Dvs, den räknar ned (-1) vid subrutinanrop och avbrott och upp med ett (+1) vid återhopp. Instruktioner som lägger värden på eller hämtar värden från stacken orsakar samma sak.

Adressavkodning och I/O

Adressavkodningen syftar till att komma åt processorns externa delar (tangenbord, spriteregister och bildminne) med samma instruktioner som använder dataminnet. Dvs, om en skriv- eller läsinstruktion använder sig av en adress som ligger utanför dataminnet så kommer adressavkodaren, beroende på adress, att aktivera dessa externa delar. Exakt vilka adresser som kommer att användas till detta bestäms under själva utvecklingen av processorn.

Tangentbordsavkodare

Tangentbordsavkodaren kommer att avkoda PS/2-tangentbordets signaler (`PS2_CLK` och `PS2_DATA`), lagra tangentbordsdatat i det speciella registret `KBD_DATA`, samtidigt sätta avbrottsvippan `IRQ` som därefter kommer att orsaka ett avbrott.

Avbrottsfunktionalitet

Avbrottsvippan IRQ kommer att orsaka ett avbrott genom att simulera ett subrutinanrop. Skillnaden blir att vid ett avbrott kommer hoppadressen från ett hårdkodat värde (1). För övrigt händer samma saker som vid ett subrutinanrop, dvs återhoppsadressen (värdet hos PC) sparas på stacken och stackpekaren SP räknar ned (-1).

Avbrottet får inte avbryta en pågående instruktion, vilket inte är nåt problem för instruktioner som tar en klockcykel, men eftersom hopp kommer att ta två klockcykler skulle ett hopp kunna avbrytas mitt i hoppet (dvs precis före hopp-NOP:en). Detta förhindras genom att muxen M3 har en viss prioritetsordning. Först i prioritet kommer alltså

hopp-NOP:en (om den ska köras), därefter själva avbrottet (om det är aktiverat via IRQ-vippan) och sist själva instruktionen från programminnet.

Spriteregister

Spriteregistren kommer att innehålla koordinatdata (X och Y) för de sprites som applikationen använder. Antalet spriteregister är obestämt. Men då dessa är adressavkodade är det lätt att lägga till fler. Spriteregistren kommer att läsas av VGA-motorn som sedan ritar ut sprites på rätt position.

Bildminne och VGA-motor

Bildminnet utgörs av ett s k block-RAM, viket möjliggör att processorn kan skriva information till minnet samtidigt som VGA-motorn kan läsa samma information utan av någon av dessa enheter behöver ta hänsyn till varandra. Bildminnet organiseras som ett rutnät om 40x30 rutor, där varje ruta kan innehålla ett tilenummer (en byte). Totalt krävs alltså inte mer än 1200 byte block-RAM.

VGA-motorn innehåller själva tileminnet (dvs utseendet på alla tiles) och ritar kontinuerligt ut tiles och sprites genom att skapa nödvändiga bild- och synksignaler.

Kollisionshantering

I nuläget är tanken att kollisionshantering mellan tiles och sprites ska ske mjukvarumässigt. Dvs programmet får kontinuerligt jämföra positioner mellan olika objekt för att veta om kollision har inträffat.

Minnesåtgång

Vi räknar med att programminnet inte behöver vara större än 1024 rader. Detta kommer att implementeras i form av LUT:ar då vi behöver kunna läsa kombinatoriskt ur programminnet. De första adresserna utgörs av reset- och avbrottsvektor(er):

```
$000 RJMP MAIN
$001 RJMP INT_KBD
```

Dataminnet tror vi inte behöver vara större än 256 rader, då det bara ska användas för temporärdata av register och diverse variabler. Även dataminnet behöver läsas kombinatorisk och implementeras därmed som LUT:ar.

Minnesmappade I/O-delar, såsom KBD_DATA, Sprite-register och bildminne får sina adresser någonstans efter dataminnet. Bildminnet behöver vara ett block-RAM (för samtidig läsning och skrivning) och ska inte kräva mer än 1200 byte för att representera en tile-yta om 40x30 tiles.

Instruktionsuppsättning

Vi tänker oss att följande instruktioner kommer att behövas:

Mnemonic	Funktion
`NOP`	No operation
`RJMP` *offset*	Hopp till PC+1+*offset*
`JSR` *offset*	Subr.anrop till PC+1+*offset*
`RET`	Återhopp från subrutin
`IRET`	Återhopp från 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 $\oplus$ V = 0
`BLT` *offset*	Hopp om N $\oplus$ 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)

Milstolpe

Efter ungefär halva projektet ska vi kunna visa en statisk tilebaserad bild på en VGA-monitor, samt ha en fungerande simulering av CPU:n där man kan se funktionen av vissa enklare instruktioner. I/O-funktionalitet och avbrott lär inte vara implementerat vid denna tidpunkt.

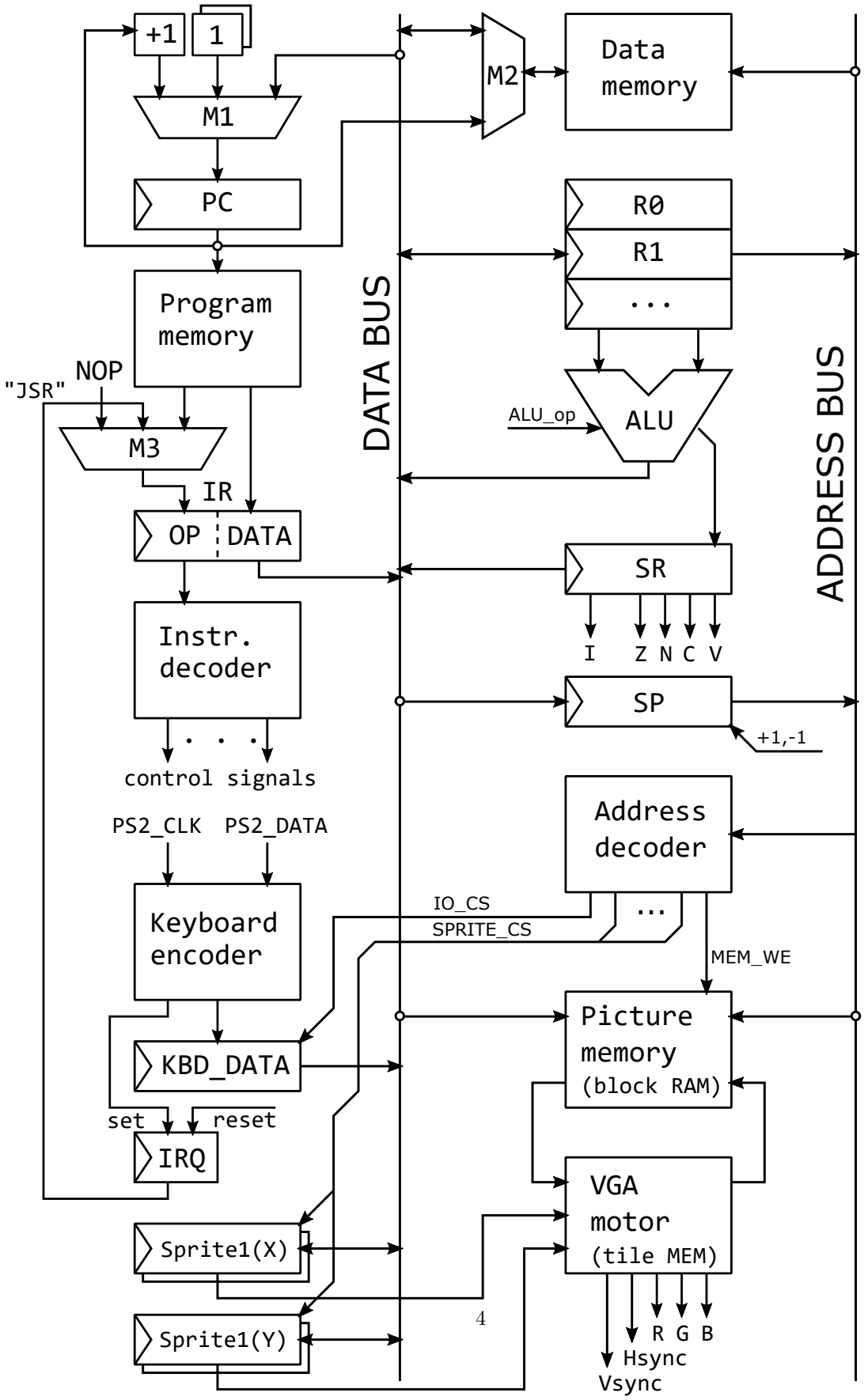

Figure 2: 2-stegs pipelinad processor med adress-avkodad I/O