DVGA03: Datorsystemteknik Datum: 07/10 - 22

Avbrott

Trafikljus

Ahmad Shammout, 020804-1293

Johanna Olsson, 020207-2443

Innehållsförteckning

[**Antaganden**](#_u2bgd8c25su7) **3**

[**Översikt**](#_ky4posesh1tr) **3**

[**Detaljerad beskrivning**](#_r26lmgrp6t5t) **5**

[**Problem**](#_xe9hynrf0a51) **6**

[**Sammanfattning**](#_fflxzpigdb84) **7**

[**Referenslista**](#_ovv27fq0p3a3) **7**

[**Bilagor**](#_66v0m9gc1afu) **7**

# Antaganden

Under projektets gång har vi gjort vissa antaganden för att slutföra laborationen. De antaganden som vi gjorde var bland annat att avbrott från knapptryck endast skulle registreras då programmet körde igenom *loop-*delen i koden. Vi antog då också att *loop* måste köras igenom ett antal gånger, det vill säga att vi ökade timern till ett högre tal, innan koden fortsätter exekveras för att användaren ska ha möjlighet att hinna trycka på en av knapparna. Utifrån laborations specifikationen antog vi även att knappen för bilar inte behöver användas då bilvägen alltid ska vara grön, så vidare ingen vill gå över övergångsstället och har tryckt på knappen för att ge övergångsstället grönt ljus.

# Översikt

I MIPS kan avbrott hanteras lite olika beroende på vilken typ av MIPS arkitektur som körs. Under laborationen har MIPS32 arkitekturen använts, men i denna rapport beskrivs avbrotts hanteringen i MIPS R3000.

I MIPS R3000 hanteras avbrott genom CP0, system control coprocessor, som finns i centralprocessorn (Naoya et al., 2017). I CP0 finns tre register till hjälp för att hantera avbrott och de är EPC-registret, Cause-registret och Status-registret. EPC-registret håller i återhoppsadressen från den platsen i koden där avbrottet skedde, Cause-registret håller i information gällande varför avbrottet skedde, och Status-registret hanterar hur avbrottet ska utföras. När en avbrottssignal skickas från en extern enhet till CP0 så hanteras det genom att återhoppsadressen sparas ned till EPC-registret samt information om avbrottet skickas till Cause-registret. Status-registret uppdateras så att systemet körs i kärnläge, kernel mode, och så att andra avbrott hindras under tiden. Sedan skickar CP0 en avbrottsstartsignal till centralprocessorn och skriver in startadressen från avbrottshanteraren till PC, programräknaren, så att centralprocessorn kan hoppa till avbrottshanteraren. Avbrottshanteraren sparar ned värden från de allmänna registren till primärminnet och kallar på rätt avbrottsrutin beroende på avbrottsorsaken. När avbrottsrutinen har genomförts så återställs de allmänna registren, exekveringsläget och förhindrandet av andra avbrott. Adressen i EPC-registret skrivs tillbaka till programräknaren, och då kan centralprocessorn fortsätta med exekveringen av programmet (Naoya et al., 2017).

Programmet ska styra ett trafikljus vid ett övergångsställe, där trafikljuset alltid ska vara grönt för bilvägen om det inte finns några fotgängare. För att kunna avgöra om det finns fotgängare så finns det en knapp att trycka på för att skicka signaler till trafikljuset att någon väntar på att övergångsstället ska få grönt ljus. Samtidigt finns det magnetslingor som känner av om det finns bilar på vägen. Trafikljuset ska lysa grönt för bilar i minst tio tidsenheter innan det kan ändras till rött via knapptryck. När ljuset är rött på bilvägen så har fotgängare tio tidsenheter på sig att ta sig till andra sidan innan ljuset blir grönt för bilvägen igen. Ljuset för fotgängare lyser grönt i sju tidsenheter för att sedan blinka med röd gubbe i tre tidsenheter för att varna fotgängare att tiden kommer att ta slut och att det snart blir rött ljus på övergångsstället. Trafikljuset för bilen växlar till gult innan den går över till röd eller grön, och det gula ljuset är kvar i tre tidsenheter.



Figur 1: Trafikljuset som ska konstrueras. Grönt för bilar, rött för fotgängare.



Figur 2: Grönt för fotgängare, rött för bilvägen.



Figur 3: Gult ljus på bilvägen under övergångarna mellan rött/grönt ljus.

# 

# Detaljerad beskrivning

I koden för laborationen ingår flera olika variabler, adresser, funktioner och annat viktigt som gör att programmet körs ordentligt. Här beskrivs viktiga detaljer i koden som ger mer förståelse till programmets funktioner. Allt som nämns i denna del hänvisas till bifogad bilaga.

I början av koden definieras de typer av avbrott som kommer att ske i programmet, vilket är tidsavbrott och avbrott från knapptryck. Detta skedde genom att deklarera dem med ett namn, samt adressen. För att aktivera knappar och timer behövde man tilldela dem vissa adresser, och även färgerna i trafikljuset tilldelades adresser i början av koden. Under .*data* definierades de strängar som skrivs ut när det trafikljuset ändrar färg. Timern som räknar tidsenheterna deklarerades även och fick värdet noll. Sist definieras en plats genom .*space* variabeln för att lagra in värdet som visar vilken status trafikljuset har. Därefter skrevs avbrottsrutinens adress, vilket är 0x80000180, samt vilken avbrottsrutin som ska exekveras när ett avbrott sker. Den avbrottsrutin som körs när ett avbrott registreras är *int\_routine.*

Den första funktionen i koden är *main*. I funktionen *main* sker mestadels förberedelser, bland annat introduktion av timern, statusregister för knappens avbrottshantering samt hur trafikljusen ska lysa vid uppstart av programmet.

Därefter kommer funktionen *loop* som är den del i koden där programmet är mottagligt till att registrera knapptryck från användaren. Funktionen låter trafikljusen behålla nuvarande tillstånd och invänta instruktioner för eventuell förändring. *Loop* körs igenom ett antal gånger beroende på ett värde på en timer i trafikljussimulatorn, om ingen förändring sker så hoppar den ut och fortsätter räkna timern för tidsenheter, för att sedan återgå till *loop* igen. Om avbrott registreras, det vill säga att knappen för fotgängare har blivit nedtryckt, när programmet är i *loop* funktionen så hoppar programmet vidare till koden för att förändra trafikljusen så att fotgängare får grönt ljus, men enbart om bilvägen har haft grönt ljus i minst tio tidsenheter.

I funktionen *int\_routine* allokeras först minne i stacken för att spara status register, passerade argument samt återhoppsadress. Sedan flyttas värdet i *cause* till *k1* och adderas med adressen till tidsavbrottet (*EXT\_INTIME*) och sparas i *k0*, denna operationen skapar tidsavbrottet. Sedan laddas värdet som finns i timern och adderas med 1, och skrivs ut med ett mellanslag. Detta för att visa varje räknad tidsenhet.

Efter *int\_routine* kommer fyra kortare funktioner. Funktionen *control* ser till att registrerade knapptryck inte går igenom förens bilvägen har haft grönt ljus i tio tidsenheter. Om knappen för fotgängare trycks ned innan tio tidsenheter har passerat så skickas programmet vidare till *returnfunc* som använder återhoppsadressen från avbrottet till att därefter skicka programmet tillbaka till *loop. Button* kollar av vilket typ av knapptryck som har skett, men då endast knappen för fotgängare fungerar så är det enbart det knapptrycket som *button* kollar efter. När *button* registrerat ett knapptryck för fotgängare så skickas programmet vidare till funktionen *clean* som återställer tiden i timern till noll, samt printar ut att det nu blir gult ljus på bilvägen. Funktionen *clean* återkommer ett antal gånger i koden i lite olika former, *clean\_1, clean\_2, clean\_3,* och *clean\_4* som sköter återställande av tiden och hopp till rätt efterföljande funktion beroende på hur trafikljusen ska förändras.

Funktionerna *yellow* och *yellow\_1* är också väldigt lika varandra. Den förstnämnda hanterar gult ljus på bilvägen för att sedan övergå till rött, den ser även till att det gula ljuset lyser i tre tidsenheter. Den andra versionen av *yellow, yellow\_1,* har samma funktion fastän med en liten skillnad. Skillnaden är att *yellow\_1* ska sköta det gula ljuset på bilvägen när det ska övergå till grönt efteråt.

Funktionerna *walk\_green* och *car\_green* är också ganska lika varandra, men den ena tar hand om gröna ljuset till fotgängare, medans den andra sköter jobbet till att det blir grönt för bilar. Skillnaden mellan dem är att *walk\_green* lyser grönt för fotgängare i sju tidsenheter sedan börjar det att blinka med röda gubbar i tre tidsenheter, men denna operationen sker i *wink* funktionen. Däremot har bilarna prioritet för ett grönt ljus i tio tidsenheter, det vill säga att bilarna ska ha ett grönt ljus i tio tidsenheter, efter tio tidensenheter kan ett avbrott från knappen gå igenom.

Den sista funktionen i koden är *restore*. Denna funktion anropas ett antal gånger när programmet körs, detta för att återställa diverse register som kan ha använts i vissa andra funktioner. Funktionen *restore* återställer även knappen och timern i Cause-registren.

# Problem

I början av laborationen var det svårt att förstå hur avbrott fungerar och hur avbrott hanteras i MIPS-processorn. Genom att testa att implementera olika kod och provköra exempelkod för avbrottshantering som vi fick av läraren, samt läsa på om avbrottshantering i MIPS, lyckades vi förstå hur avbrott skedde och hur de skulle implementeras i laborationen.

Därefter var det svårt att implementera kod som fungerar och hänger ihop mellan timer och knapptryck. Knapptryck var inte svårt att implementera, men däremot var det svårare att skapa en timer som sätter villkor på hur länge trafikljuset ska lysa vid varje ljusförändring. Samtidigt skulle timern återställas och börja att räkna om från noll varje gång ljuset ändrades. För att lösa detta kollade vi på exempelkod gällande timer som vi har fått av läraren, samt kollade tillbaka på kod från tidigare laboration i MIPS för att få en idé om hur diverse if-satser eller villkor kunde skrivas för att begränsa knapptryck och implementera timern på ett bra sätt.

Det var också ett problem att hinna trycka på knappar när koden var i *loop*, detta på grund av att loopen körde igenom koden väldigt snabbt. Problemet löstes genom att ändra tiden på timern i traffic light simulator till ett hög siffra t.ex 100, samtidigt kunde man ändra från full run speed till ett lägre värde så att koden inte exekverades för snabbt.

Det vi lärde oss av problemen under laborationens gång var att det är väldigt viktigt att hela tiden överblicka den kod som är skriven, ibland kan problemen ligga i delar som man egentligen inte har förväntat sig. Det är också väldigt viktigt att ha bra kunskap om hur alla delar i programmet fungerar och hur de påverkar varandra, till exempel när det var svårt att hinna trycka på knapparna när programmet befann sig i *loop* endast en kort stund. I ett sådant läge behöver man ha förståelse för hur alla delar i programmet fungerar och det var inte förens vi läste på ordentligt om avbrottshantering som vi förstod hur problemet skulle lösas.

# Sammanfattning

Projektet med trafikljus simulatorn har gått ganska smidigt och bra. Det fanns mycket bra tips och hjälp på Canvas från läraren som vi använde oss av under laborationens gång. Vi har fått en mycket bättre förståelse för hur avbrott fungerar och hur man kan använda de i praktiken, det vill säga när man skriver kod. Laborationen har varit ett väldigt bra hjälpmedel till att få en djupare förståelse för flera av kursens moment, och gett oss möjligheten att repetera vissa moment som var med under tidigare laboration. Koden som vi har skrivit för att slutföra laborationen är enligt oss inte perfekt, det finns vissa saker som är möjliga att förbättra, men som för oss kändes lite för svåra att förstå i det här läget. Vi använde oss till exempel av vissa funktioner flera gånger med liknande namn tex. *clean* och *yellow* funktionerna, där skillnaden mellan funktionerna enbart var en rad med kod. Detta gjorde vi dock för att det annars skulle bli lite mycket hopp upp och ned i koden under programmets körning, och svårt för oss att hänga med och göra hoppen på ett snyggt sätt.

Ungefärlig tidsåtgång för att skriva koden för ändring av färg i trafikljusen är tre timmar. Tidsåtgången för att skapa timern och göra diverse begränsningar med hjälp av denna tror vi ligger på närmare fyra eller fem timmar.

# Referenslista

Naoya, I., Yuuki, O., Nagisa, I., Hiroyuki, K., & Hiroyuki, T. (Oktober 2017). Binary Synthesis Implementing External Interrupt Handler as Independent Module. Association for Computing Machinery. Hämtat från <https://dl.acm.org/doi/pdf/10.1145/3130265.3130317>

# Bilagor