Maskinnära Programmering

When life gives you lemons, make orange juice and leave the world wondering how the hell you did it.

# Del 1

## Översätta Hex, Dec och Bin

## Bitwise AND, OR, XOR, NOT

0101 (decimal 5)

AND 0011 (decimal 3)

= 0001 (decimal 1)

0101 (decimal 5)

OR 0011 (decimal 3)

= 0111 (decimal 7)

0101 (decimal 5)

XOR 0011 (decimal 3)

= 0110 (decimal 6) (Disallows doubles)

NOT 0111 (decimal 7)

= 1000 (decimal 8) (Inverts)

## Binär tvåkomplementsform

Om talet börjar på 1 (sign 1 = negativ), invertera alla bitar, negera och addera 1.

1111 1111 ->   
 0000 0000 ->   
-0000 0000 ->   
-0000 0001 ->   
-1

## IEEE-754 standard för decimaltal

S (1 bit) EXPONENT (8 bits) SIGNIFICAND (23 bits)

(single precision, double uses 11 bit exponent, 52 bit significand)

1.SIGNIFICAND \* 2^EXPONENT

Exempel:

1 10000101 11011010100000000000000  
  
1 = negative  
1000 0101 = 133, unbias by sub 127: 133 – 127 = 6  
1101 1010 1 = normalize: 1.1101 10101 \* 2^6 (six from unbiased exponent!)  
  
We move point six steps as the exponent is 6  
1 (negative) 1110110.101 (binary point)  
= -118.625 (dec)

## Talnormalisering

Flytta decimalpunkt i enlighet med en exponent.  
01 . 1010 \* 2^-2  
0110 . 10

## Uttyda instruktioner ur hexadecimal byte

6 bitar operationskod, I-, J- och R-typ  
5 bitar register  
16 bitar konstanter  
26 bitar adresser

# Del 2

## Processorns pipeline

1. Instruction fetch – hämta instruktionen från minnet
2. Instruction decode & register fetch – Avkoda instruktionen och hämta operanderna från relevanta register (simultant i MIPS)
3. Execution – Exekvera operationen eller beräkna den önskade adressen
4. Memory access – Läs data från minnet
5. Writeback – Skriv data till minnet

Kan jobba parallellt eller i cykler. Olika typer av konflikter kan uppstå. Tre vanliga pipelinekonflikter inkluderar strukturella konflikter, datakonflikter och flödeskonflikter.

## Minnesbuss

En minnesbuss är en kanal för intern överföring av data mellan komponenter som exempelvis processorn, dataminne och I/O-kontroller.

När en processor ”pollar” efter I/O utförs en iterativ kontroll av någon slags signal. En modul, exempelvis en handkontroll, kan vara kopplad till datorns I/O-kontroller och skicka en digital eller analog signal. CPU skickar då data till både adressbussen och databussen, som båda i sin tur färdas till I/O kontrollern för att hämta singalen.

Vid vanlig datahämtning/skrivning färdas data från CPU till adressbussen, som färdas till minnesenheten som i sin tur skickar data till databussen.

## Hantering av konstantvärden och hårdvaruregister (MUX)

En multiplexer (MUX) är en enhet i processorn vars uppgift är att ta emot ett två eller fler analoga eller digitala signaler, utifrån en ”select-line” välja en av dem och föra den vidare från enheten som output. Detta system är viktigt för att kunna hålla processorns design minimal och effektiv. Flera olika datapaket i olika delar av kretsen kan sammanfogas till en gemensam yta med hjälp av multiplexern. I en instruktion kan multiplexern bland annat användas för att avgöra vilket register som ska skrivas till, exempelvis rd (destination) i en R-typsinstruktion och rt (target) i en I-typsinstruktion. Den kan även välja mellan att föra vidare ett register kontra en konstant till ALU.

## Pipelinekonflikter

Om två eller fler steg i en pipeline försöker arbeta med samma komponent (t.ex ett minneslagrat värde) kan en **strukturell konflikt** uppstå. i.  
En strukturell konflikt kan uppstå när två eller fler olika steg i processorns pipeline försöker nå samma data simultant. Denna konflikt kan undvikas genom att låta data och instruktioner förvaras i olika cacheminnen. Detta kommer att kraftigt minska risken för minneskonflikter.

Om två eller fler pipelinade operationer försöker nå ett värde från en operation som ännu inte är avslutad kan en **datakonflikt** uppstå.  
En datakonflikt kan uppstå när en process eller instruktion som är beroende av data som ännu inte existerar försöker exekvera. En instruktion kan exempelvis försöka använda resultatet av en matematisk uträkning som ännu inte är färdig. Denna typ av konflikt kan undvikas genom att ge data tillräckligt med tid för att processeras, genom att justera ordningen på processerna i pipelinen eller manuellt med tomma icke-operationer (nop), vilket dock ger upphov till ”bubblor” i pipelinen.

Om en pipelinad operation försöker använda ett värde som är beroende av en branch kan en **flödeskonflikt** uppstå, då resultatet av en branch kräver en extra operation för att utföras.  
En flödeskonflikt uppstår när en instruktion är beroende av data från en tidigare instruktion, medan en branch inte kan utföras före ett exekveringssteg. Innan flödet kan fullföljas måste en delay slot tas hänsyn till. Den kan fyllas med en nop, vilket är en ineffektiv lösning, eller så kan en annan instruktion utföras i sloten i relevant eller externt syfte.

## Förklara hur en processors komponenter interagerar

* CPU – Central Processing Unit
* IFU – Instruction Fetch Unit  
  Läser instruktioner från minnet och utför avkodning av instruktionerna.
* ALU – Arithmetic Logic Unit  
  Utför aritmetiska och logiska operation på intergrer.
* FPU – Floating Point Unit  
  Utför arimetiska operationer på flyttal.
* MMU – Memory Management Unit  
  Läser och skriver data till och från minnet.

*Exempel på en operation:*

1. Instruktioner lagras (och läses) i minnet av minnehanteringsenheten (MMU).
2. Instruktionsenheten (IFU) läser och avkodar instruktioner från minneshanteringsenheten (MMU).
3. Om instruktionen gäller en operation av integrer eller logik skickas den vidare till den aritmetiska/logiska enheten (ALU).
4. Gäller det en flyttalsoperation skickas den istället till flyttalshanteringsenheten (FPU).
5. ALU och FPU hämtar resurserna för de instruerade ioperationerna från angivna register. Resultatet sparas i sin tur åter i register.

# Del 3

## I/O-hantering genom register (polling-/avbrottsbaserad)

Input-/Output kan bland annat registreras och hanteras baserat på två system; polling och avbrott.

**Polling** innebär att CPU:n iterativt ”lyssnar” efter en signal. Om den hittar en signal registreras och hanteras den. Signaler som skickas medan CPU:ns loop är i ett annorlunda steg kommer inte registreras, vilket kan vara ett problem i flera realtidsinteraktioner.

**Avbrottsbaserad I/O** innebär att när en I/O-signal skickas så kallar den CPU:ns uppmärksamhet, oavsett vad den pysslar med för tillfället. Detta innebär mer träl för processorn men mer flexibel I/O-hantering i enhetens avseende. Denna typ av avbrott kallas för *externa avbrott.*

I MIPS hanteras ett avbrott genom att den nuvarande exekveringen avbryts och *Cause*- och EPC-register uppdateras. Cause innehåller information om vilken typ av avbrott som skett. EPC (Exception Program Counter) håller en adress till vilken flödet ska återvända efter att avbrottshanteringen avslutats. Programflödet hoppar sedan till adressen 0x80000080 som innehåller en rutin för avbrottshantering. I vanliga fall läser denna rutin av Cause- och EPC-registerna och kallar i sin tur på specifika hanterare.

## Externa avbrott

## Cache (associativitet, mappning, nivåer, replacement policy)

Cache – en säker förvaringsplats. Minne nära processor för snabb tillgång.

Byggs med Static RAM-struktur, vilket är dyrt men resurssnålt och oerhört snabbt. Jobbar direkt mot processorn, hämtar sin data från DRAM som hämtar sin data från det sekundära minnet, t.ex en hårddisk.

Ett cacheminne där varje minnesplats är mappat till exakt en plats i cachen kallas för en **direktmappad cache**. I motsats kan en **fullt associativ cache**lagra datablock vart som helst. Andra strukturer hänvisas vanligtvis till som **set-associativa**. Rent statistiskt har fullt associativa cachar betydligt lägre miss rate än direktmappade cachar, däremot är deras söktid längre och hårdvara dyrare. Allt utom fullt associativ cache kräver en *replacement policy*.

En replacement policy innebär en algoritm som avgöra vilka element i en cache som bör ersättas för att ge plats åt ett nytt element. En vanligt förekommande replacement policy är LRU (Last Recently Used) som väljer ut det äldsta använda elementet i cachen.

## Seriell/parallel kommunikation

Seriell kommunikation innebär att datasignaler skickas en bit i taget. Parallell kommunikation innebär att ”paket” med hela bytes skickas och tolkas en bit i taget.

**Parallell kommunikation** är väldigt snabb, då en flera trådar parallellt kan skicka bitar. Dock är parallella signaler betydligt känsligare för störningar, vilket gör det till en instabil lösning över längre sträckor.

**Seriell kommunikation** använder endast tre trådar; en var för överföring i vardera riktning samt en tråd för jordning. Den resulterande signalen är stabil och pålitligare än den hos en parallell I/O-enhet.

## Data Direction Registers

## Tidslokalitet/Rymdlokalitet

## Direct Memory Access

# Del 4

## Översätt C-kod

## Stack

## Logical/Artihmetic shift

## Syntetiska instruktioner

## Selektion

## Aktiveringsblock