UPPSALA UNIVERSITET

Informationsteknologi

Today's class

- Finish computer system overview
- Review of more C

Finish computer system overview

UPPSALA UNIVERSITET

Informationsteknologi

Instruction Execution

- Two steps
  - Processor reads (fetches) instructions from memory
  - Processor executes each instruction

Instruction Cycle

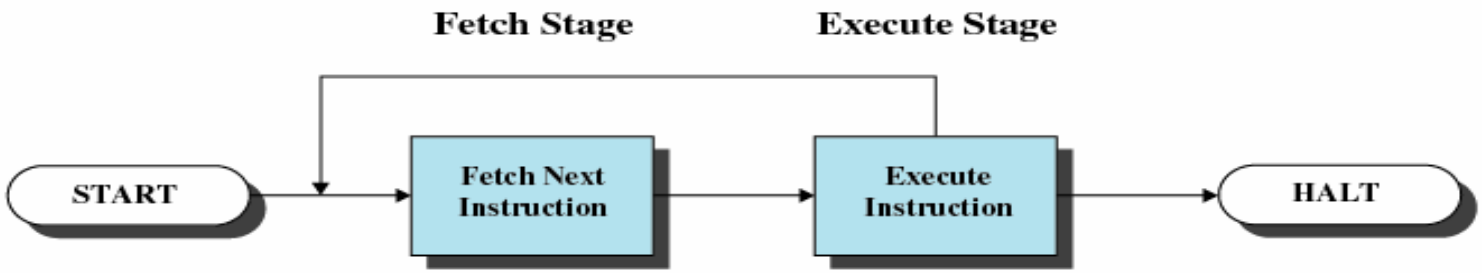


Figure 1.2 Basic Instruction Cycle

UPPSALA UNIVERSITET

Informationsteknologi

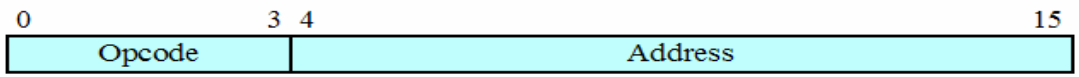
Instruction Fetch and Execute

- Program counter (PC) holds address of the instruction to be fetched next
- The processor fetches the instruction from that memory location
- Program counter is incremented after each fetch

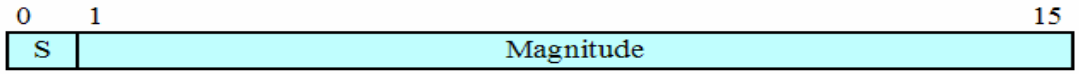
Instruction Register

- Fetched instruction is placed in the instruction register
- Categories
  - Processor-memory
    - Transfer data between processor and memory
  - Processor-I/O
    - Data transferred to or from a peripheral device
  - Data processing
    - Arithmetic or logic operation on data
  - Control
    - Alter sequence of execution

Characteristics of a Hypothetical Machine



(a) Instruction format



(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Example of Program Execution

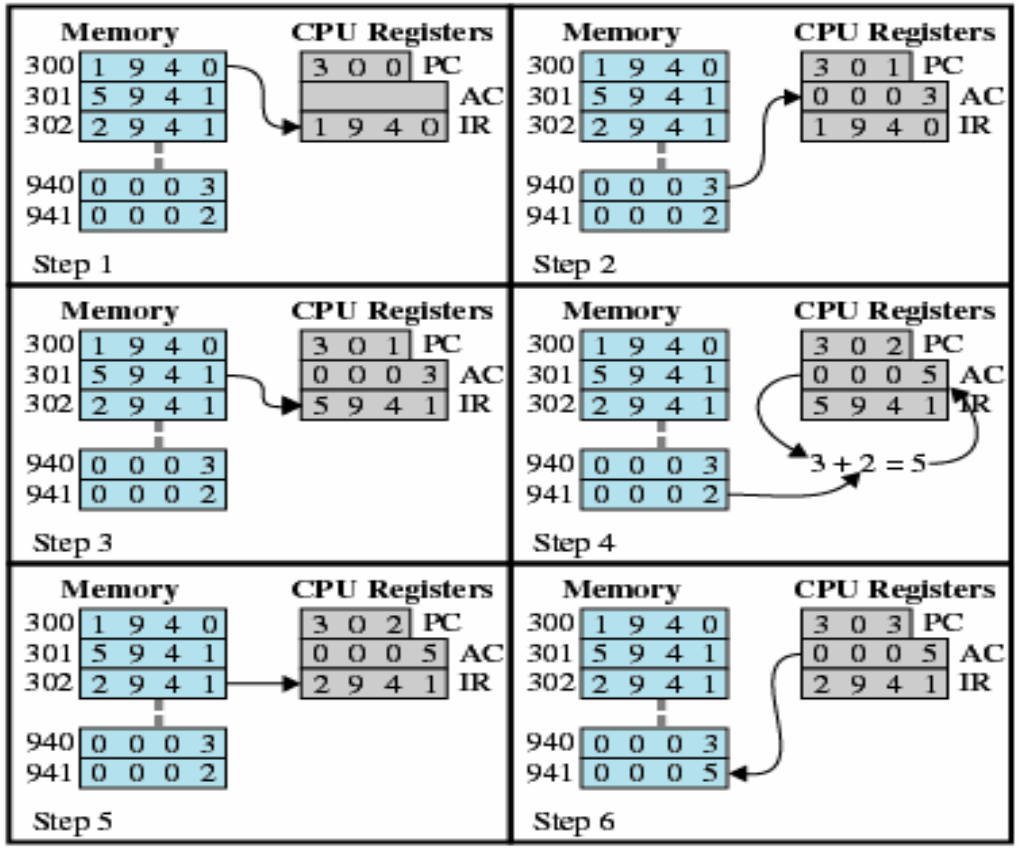


Figure 1.4 Example of Program Execution
(contents of memory and registers in hexadecimal)

Direct Memory Access (DMA)

- I/O exchanges occur directly with memory
- Processor grants I/O module authority to read from or write to memory
- Relieves the processor of the responsibility for the exchange

UPPSALA UNIVERSITET

Informationsteknologi

Interrupts

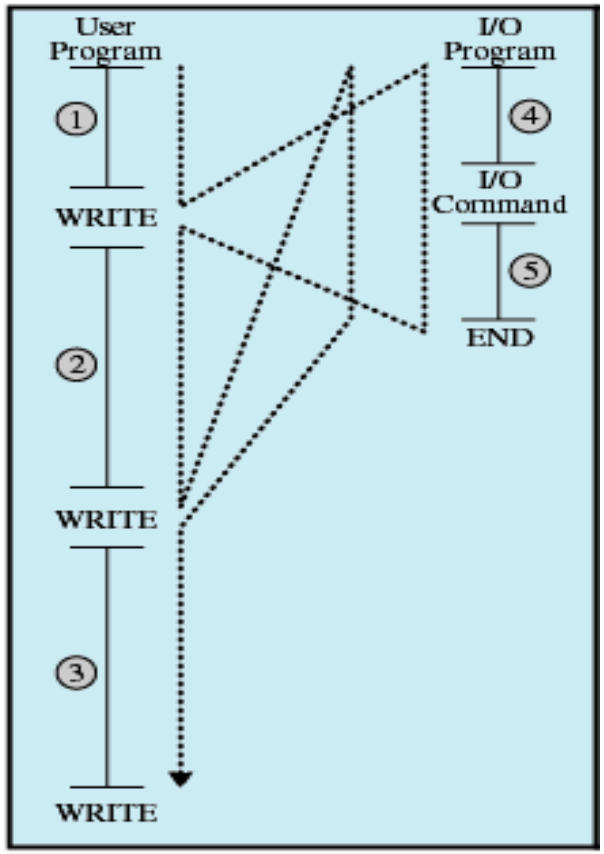
- Interrupt the normal sequencing of the processor
- Most I/O devices are slower than the processor
  - Processor must pause to wait for device

Classes of Interrupts

Table 1.1 Classes of Interrupts

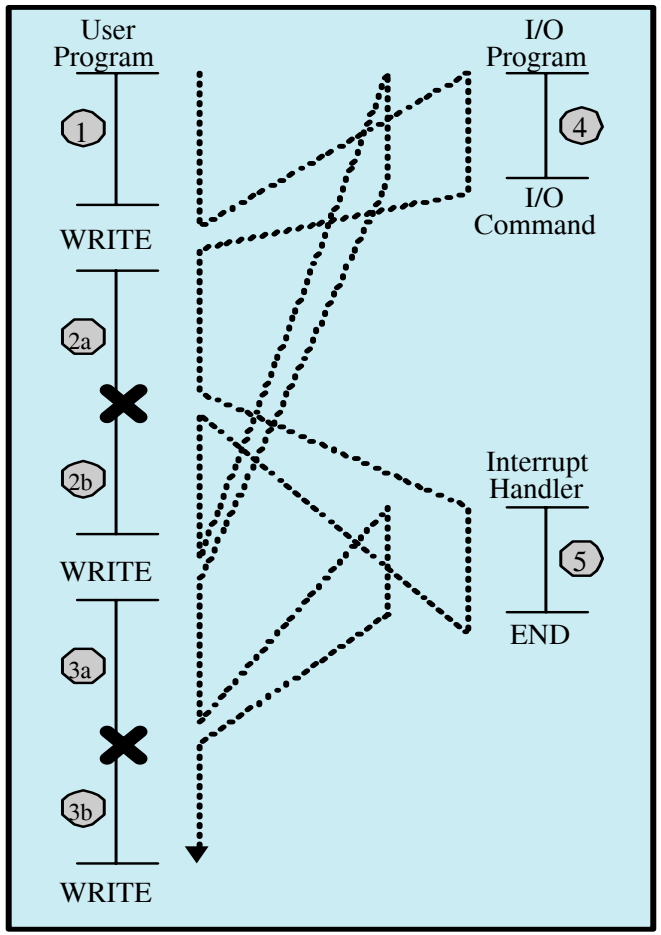
Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Program Flow of Control Without Interrupts



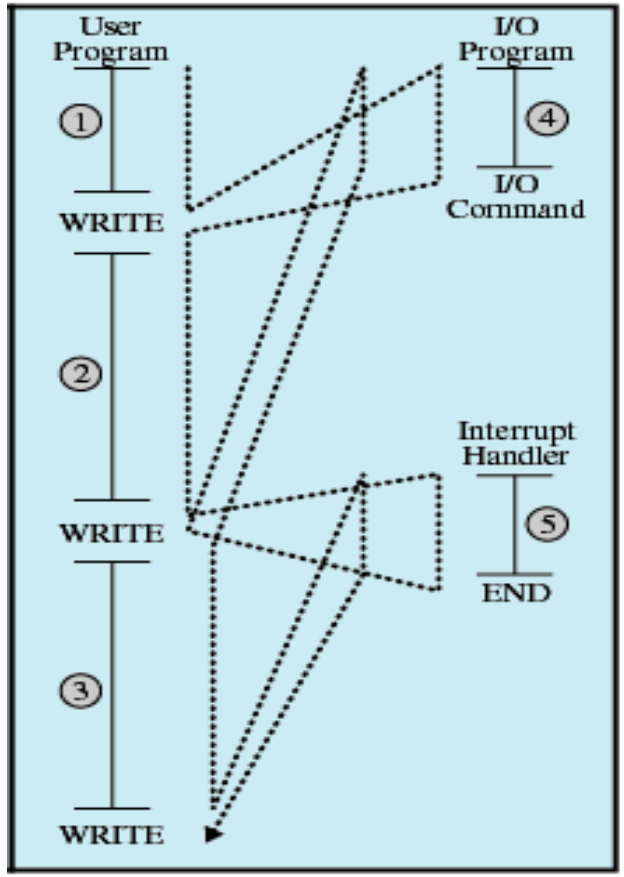
(a) No interrupts

Program Flow of Control With Interrupts, Short I/O Wait



(b) Interrupts; short I/O wait

Program Flow of Control With Interrupts; Long I/O Wait



(c) Interrupts; long I/O wait

Interrupt Handler

- Program to service a particular I/O device
- Generally part of the operating system

Interrupts

- Suspends the normal sequence of execution

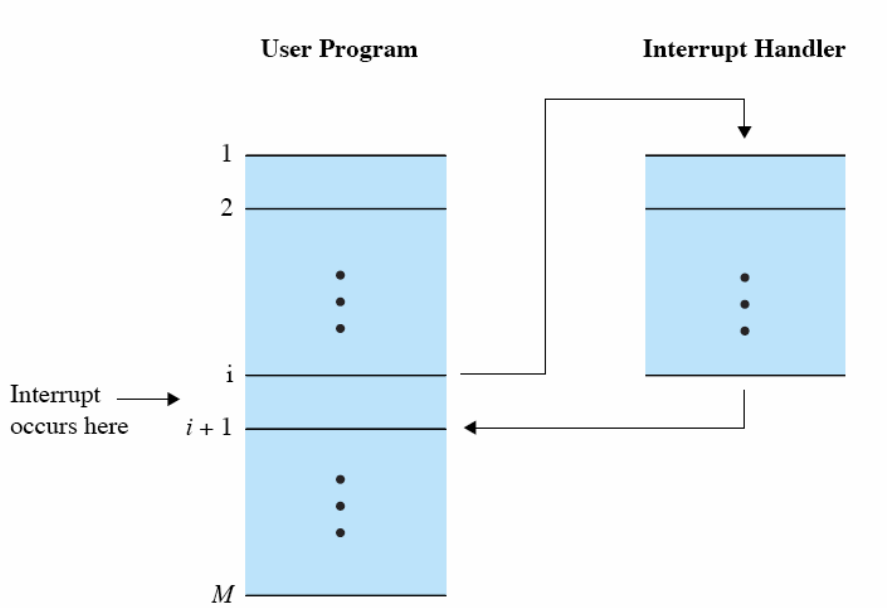


Figure 1.6 Transfer of Control via Interrupts

Interrupt Cycle

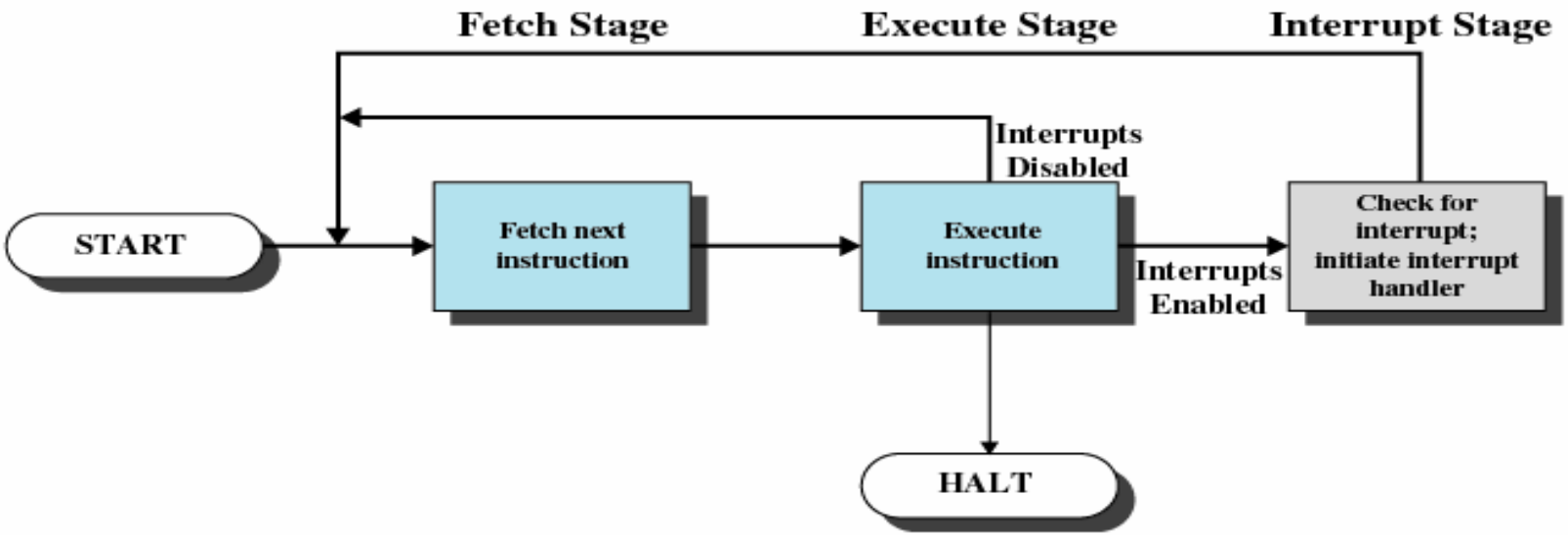


Figure 1.7 Instruction Cycle with Interrupts

UPPSALA UNIVERSITET

Informationsteknologi

Interrupt Cycle

- Processor checks for interrupts
- If no interrupts fetch the next instruction for the current program
- If an interrupt is pending, suspend execution of the current program, and execute the interrupt-handler routine

Timing Diagram Based on Short I/O Wait

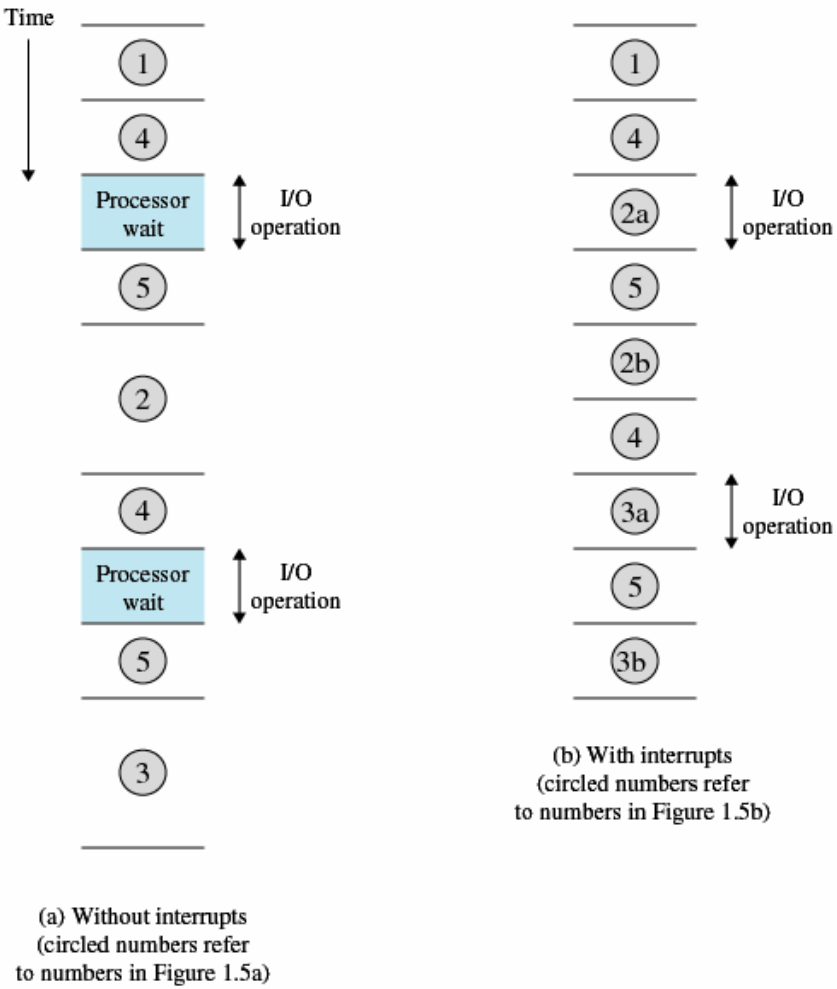


Figure 1.8 Program Timing: Short I/O Wait

Timing Diagram Based on Long I/O Wait

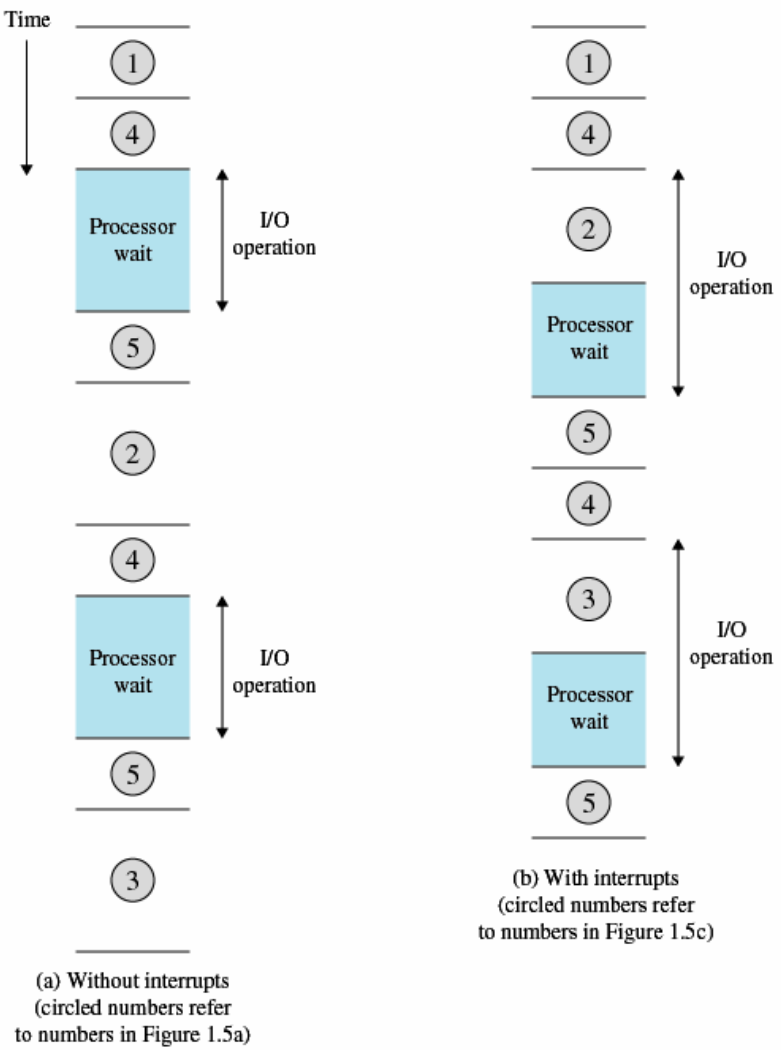


Figure 1.9 Program Timing: Long I/O Wait

Simple Interrupt Processing

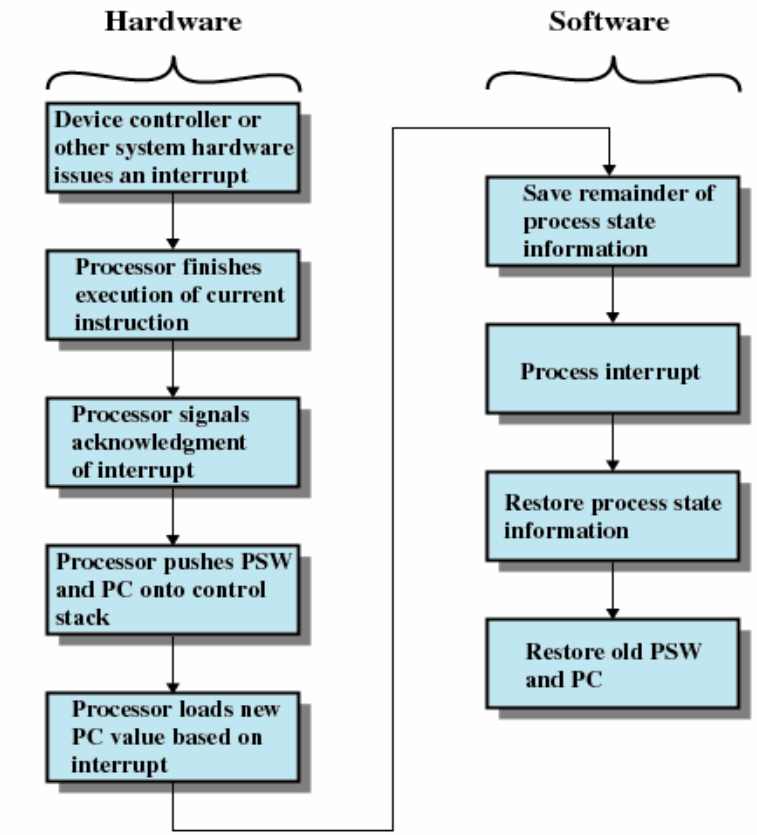
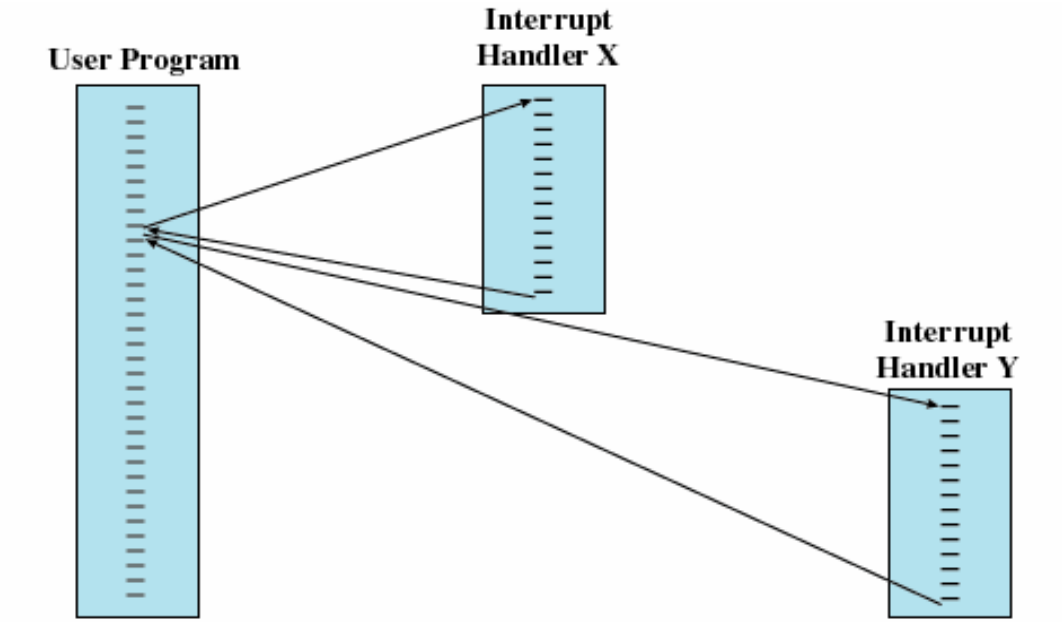


Figure 1.10 Simple Interrupt Processing

Multiple Interrupts

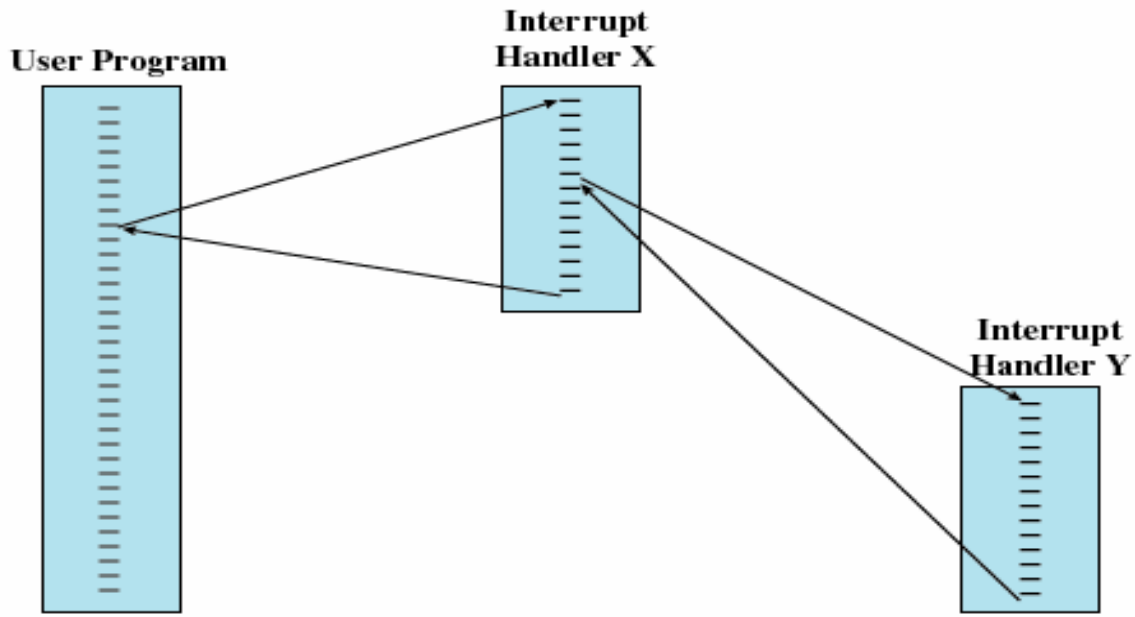
- Disable interrupts while an interrupt is being processed



(a) Sequential interrupt processing

Multiple Interrupts

- Define priorities for interrupts



(b) Nested interrupt processing

Multiple Interrupts

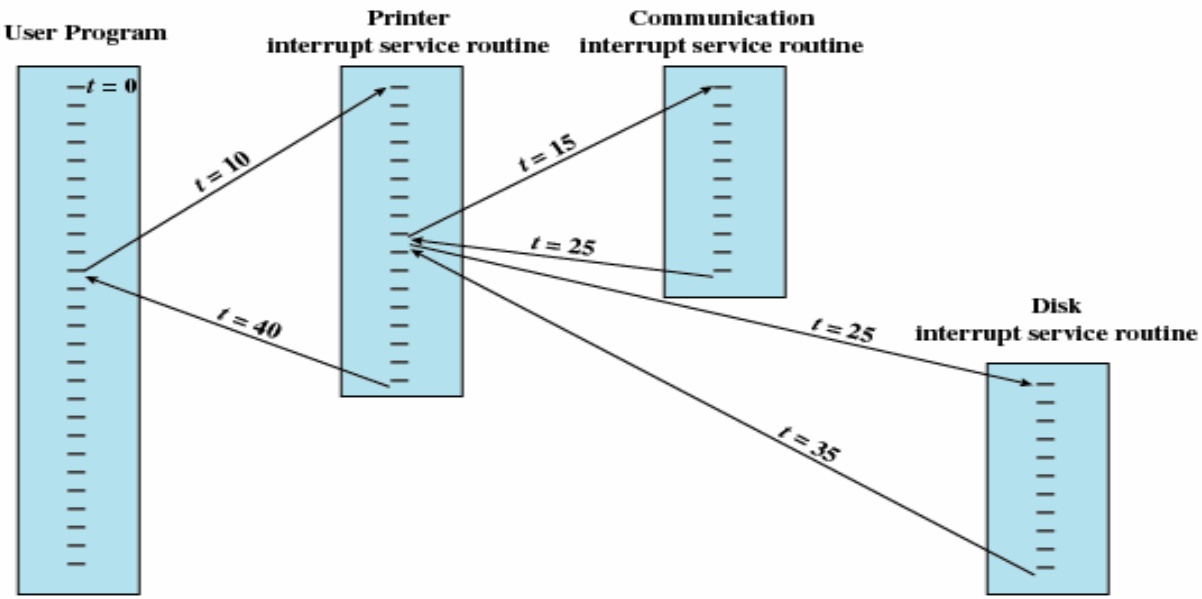


Figure 1.13 Example Time Sequence of Multiple Interrupts

Multiprogramming

- Processor has more than one program to execute
- The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
- After an interrupt handler completes, control may not return to the program that was executing at the time of the interrupt

UPPSALA UNIVERSITET

Informationsteknologi

Memory

- Faster access time, greater cost per byte
- Greater capacity, smaller cost per byte
- Greater capacity, slower access speed

Memory Hierarchy

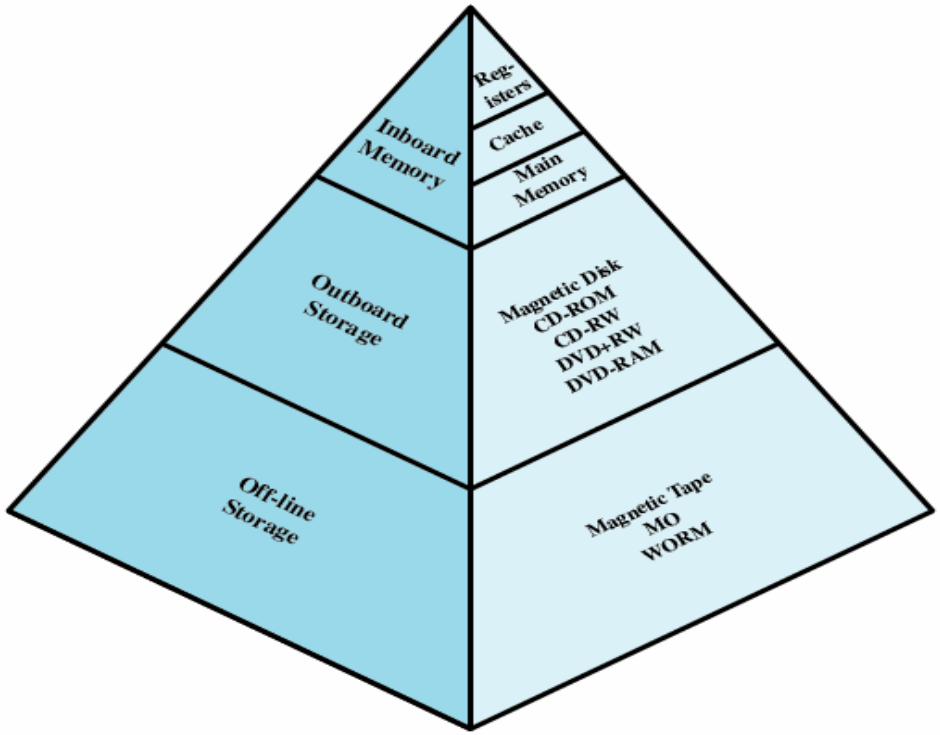


Figure 1.14 The Memory Hierarchy

Going Down the Hierarchy

- Decreasing cost per byte
- Increasing capacity
- Increasing access time
- Decreasing frequency of access of the memory by the processor
  - Locality of reference

Secondary Memory

- Nonvolatile
- Auxiliary memory
- Used to store program and data files

Disk Cache

- A portion of main memory used as a buffer to temporarily to hold data for the disk
- Disk writes are clustered
- Some data written out may be referenced again. The data are retrieved rapidly from the software cache instead of slowly from disk

Cache Memory

- Invisible to operating system
- Increase the speed of memory
- Processor speed is faster than memory speed
- Exploit the principle of locality

Cache Memory

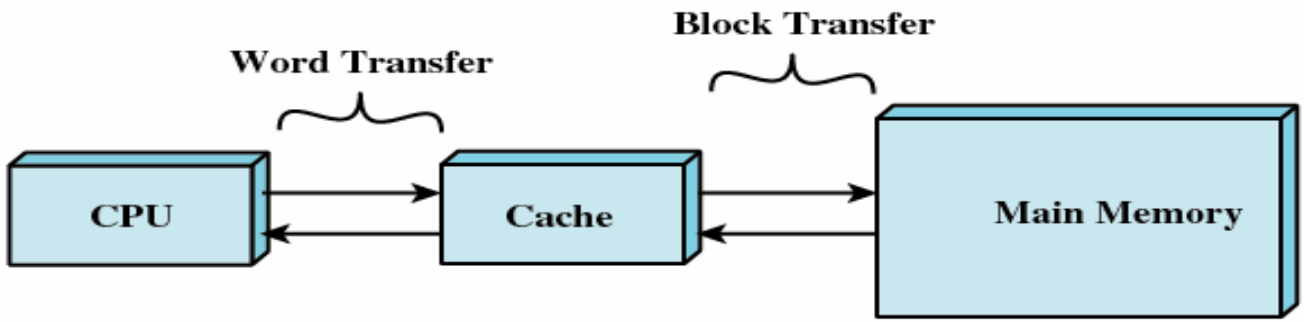


Figure 1.16 Cache and Main Memory

Cache Memory

- Contains a copy of a portion of main memory
- Processor first checks cache
- If not found in cache, the block of memory containing the needed information is moved to the cache and delivered to the processor

Cache Read Operation

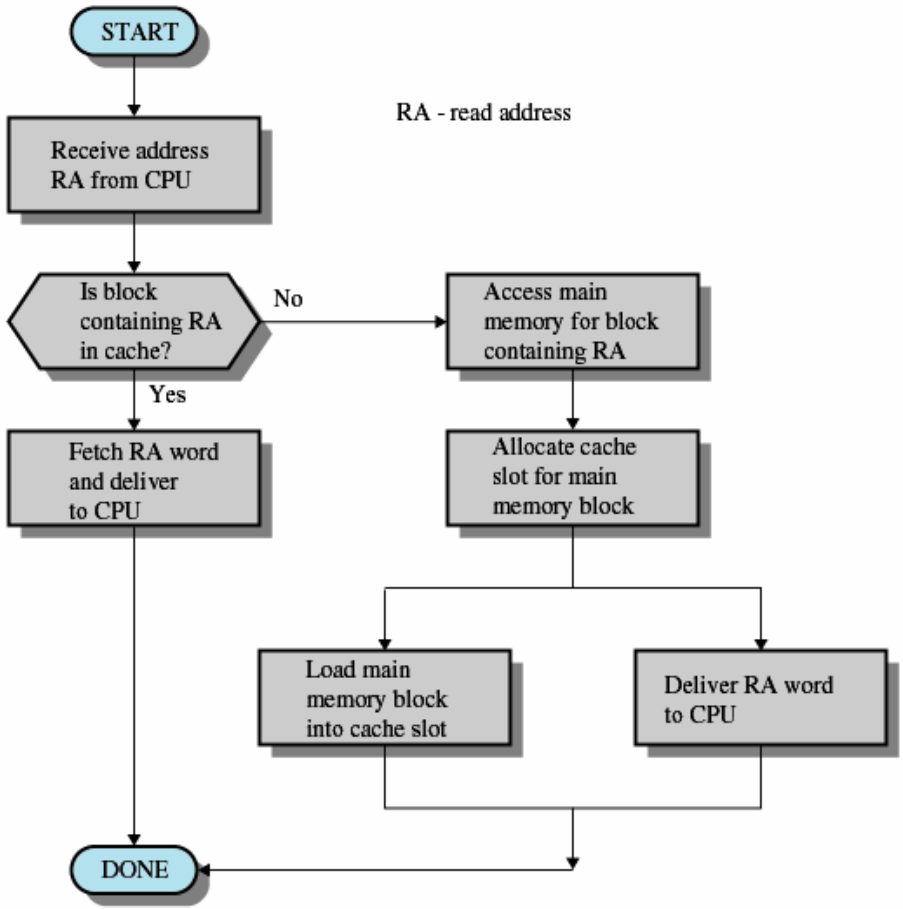


Figure 1.18 Cache Read Operation

Cache Data Modified

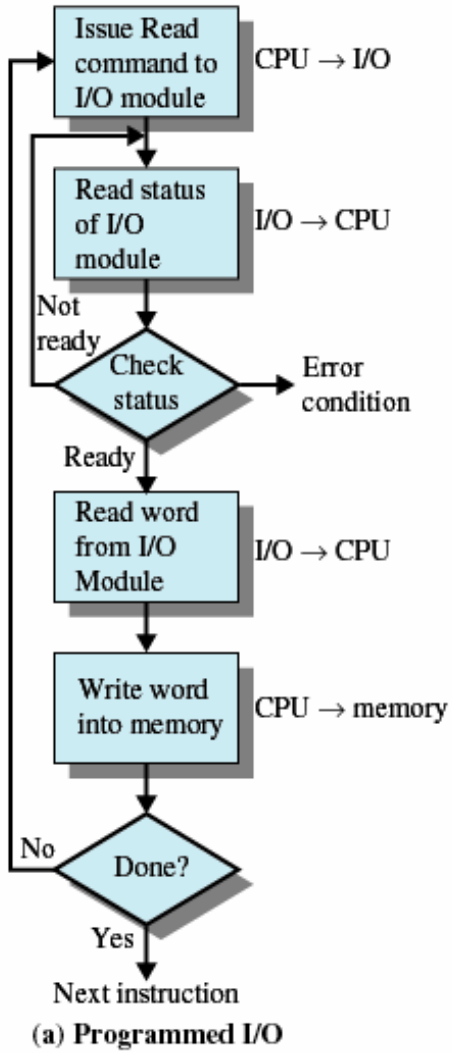
- Write policy dictates when the memory write operation takes place
  - Can occur every time cache block is updated
  - Can occur only when cache block is replaced
    - Minimizes memory write operations
    - Leaves main memory in an obsolete state

I/O Communication Techniques

- Programmed I/O
- Interrupt-driven I/O
- Direct memory access (DMA)

Programmed I/O

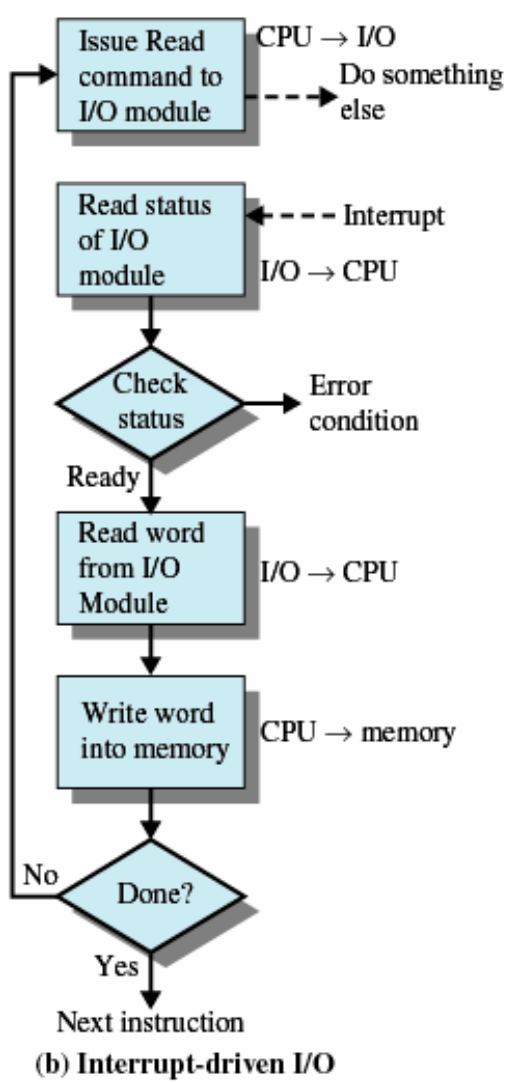
- I/O module performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete



(a) Programmed I/O

Interrupt-Driven I/O

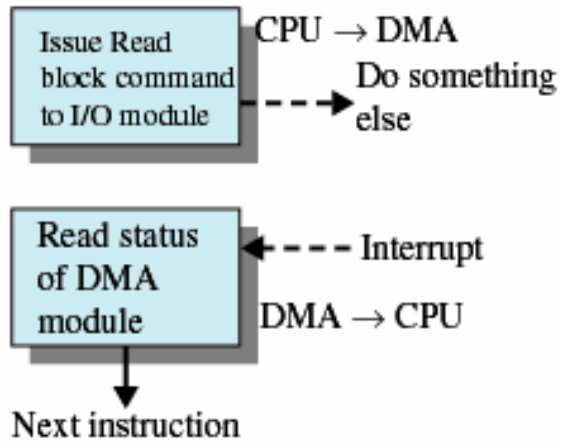
- Processor is interrupted when I/O module ready to exchange data
- Processor saves context of program executing and begins executing interrupt-handler
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor



(b) Interrupt-driven I/O

Direct Memory Access

- Transfers a block of data directly to or from memory
- An interrupt is sent when the transfer is complete
- Processor continues with other work



(c) Direct memory access

Review of more C

Structures

- Equivalent of Java's classes with only data (no methods)

```
Example 6:
#include <stdio.h>

struct birthday{
    int month;
    int day;
    int year;
  };                          //Note the semi-colon

int main() {
  struct birthday mybday; /* - no 'new' needed ! */
                          /* then, it's just like Java ! */
  mybday.day=1; mybday.month=1; mybday.year=1977;
  printf("I was born on %d/%d/%d", mybday.day,
                          mybday.month, mybday.year);
}
```

More on structures

```
struct person{
    char name[41];
    int age;
    float height;
    struct {        /* embedded structure */
      int month;
      int day;
      int year;
    } birth;
 };

struct person me;
me.birth.year=1977;……

struct person class[60];
       /* array of info about everyone in class */
class[0].name="Gun"; class[0].birth.year=1971;……
```

typedef

- typedef struct person myPerson
  - Defines a new type name myPerson as a synonym for type struct person

```
int main(){
        myPerson me;
        me.age = 6;
            …
}
```

User-defined header files

- Structures and other data structures may be defined in a header file, for better organization of the code
- These are user-defined header files e.g. person.h
- To include it:

```
#include "person.h"
```

at the start of the program file

Command line arguments

- Accept inputs through the command line.
- `main(int argc, char* argv[])`
  - `argc` – argument count
  - `argv[]` – value of each argument

UPPSALA UNIVERSITET

Informationsteknologi

Example 7

```
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int count = 0;
    if(argc < 2){
                printf("Must enter at least one argument\n");
                printf("Example: ./a.out this is program 7\n");
                exit(1);
    }
    printf(" The number of arguments is %d\n", argc);
    printf("And they are :\n");
    while(count < argc){
                printf("argv[%d]: %s\n",count,argv[count] );
                count++;
   }
   printf("\n");
   return 0;
}
```