Introduction to Threads

- Interrupts create a multithreaded environment with a single foreground thread (the main program), and multiple background threads (the ISRs).
- Projects where modules are loosely coupled, multiple foreground threads may be necessary.
- The chapter presents techniques to implement multiple foreground threads (the scheduler).
- It also presents synchronization tools, semaphores, that allow threads to interact with each other.
Thread Lists

(See Figure 5.4)

Round-Robin Scheduler

(See Figure 5.5)

Thread Control Block

- A thread control block (TCB) stores information private to each thread, and it must contain:
  1. A pointer so that it can be chained into a linked list.
  2. The value of its stack pointer.
  3. A stack area for local variables and saved registers.
- A TCB may also contain:
  1. Thread number, type, or name.
  2. Age, or how long this thread has been active.
  3. Priority.
  4. Resources that this thread has been granted.

Thread Registers

(See Figure 5.6)
### Slide 9

**C for the Threads**

```c
int Sub(int j){ int i;
    PORTC=1; /* Port C=program being executed */
    i=j+1;
    return(i);}
void ProgA(){ int i;
    i=5;
    while(1) { PORTC=2; i=Sub(i);}}
void ProgB(){ int i;
    i=6;
    while(1) { PORTC=4; i=Sub(i);}}
```

### Slide 10

**Assembly for the Threads**

<table>
<thead>
<tr>
<th>ProgA pshx</th>
<th>ProgB pshx</th>
<th>Sub pshx</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsx</td>
<td>tsx</td>
<td>tsx</td>
</tr>
<tr>
<td>ldd #5</td>
<td>ldd #5</td>
<td>std 0, x</td>
</tr>
<tr>
<td>std 0, x</td>
<td>std 0, x</td>
<td>ldaa #1</td>
</tr>
<tr>
<td>LoopA ldaa #2</td>
<td>LoopB ldaa #4</td>
<td>staa PORTC</td>
</tr>
<tr>
<td>staa PORTC</td>
<td>staa PORTC</td>
<td>ldd 0, x</td>
</tr>
<tr>
<td>ldd 0, x</td>
<td>ldd 0, x</td>
<td>addd #1</td>
</tr>
<tr>
<td>jsr sub</td>
<td>jsr sub</td>
<td>pulx</td>
</tr>
<tr>
<td>std 0, x</td>
<td>std 0, x</td>
<td>bra LoopA</td>
</tr>
<tr>
<td>bra LoopA</td>
<td>bra LoopB</td>
<td>rts</td>
</tr>
</tbody>
</table>

### Slide 11

**Thread Control Block in C**

```c
struct TCB
{
    struct TCB *Next; /* Link to Next TCB */
    unsigned char *SP; /* SP when not running */
    unsigned int Id; /* output to PortB */
    unsigned char MoreStack[49]; /* more stack */
    unsigned char CCR; /* Initial CCR */
    unsigned char RegB; /* Initial RegB */
    unsigned char RegA; /* Initial RegA */
    unsigned int RegX; /* Initial RegX */
    unsigned int RegY; /* Initial RegY */
    void (*PC)(void); /* Initial PC */
};
typedef struct TCB TCBType;
typedef TCBType *TCBPtr;
```

### Slide 12

**Thread Control Block in C**

```c
TCBType sys[3]=
{
    &sys[1], /* Pointer to Next */
    &sys[0].MoreStack[49], /* Initial SP */
    1, { 0 }, /* Id, clear stack */
    0x40,0,0,0,0, /* CCR,B,A,X,Y */
    ProgA, }, /* Initial PC */
    &sys[2], /* Pointer to Next */
    &sys[1].MoreStack[49], /* Initial SP */
    2, { 0 }, /* Id, clear stack */
    0x40,0,0,0,0, /* CCR,B,A,X,Y */
    ProgA, }, /* Initial PC */
    &sys[0], /* Pointer to Next */
    &sys[2].MoreStack[49], /* Initial SP */
    4, { 0 }, /* Id, clear stack */
    0x40,0,0,0,0, /* CCR,B,A,X,Y */
    ProgB, } ); /* Initial PC */
```
**Thread Control Block in Assembly**

<table>
<thead>
<tr>
<th>TCB1</th>
<th>TCB2</th>
<th>TCB3</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>fdb</td>
<td>fdb</td>
<td>fdb</td>
<td>fdb</td>
</tr>
<tr>
<td>IS1</td>
<td>IS2</td>
<td>IS3</td>
<td>SP</td>
</tr>
<tr>
<td>fcb 1</td>
<td>fcb 2</td>
<td>fcb 4</td>
<td>Id</td>
</tr>
<tr>
<td>rmb 49</td>
<td>rmb 49</td>
<td>rmb 49</td>
<td></td>
</tr>
<tr>
<td>fcb $40</td>
<td>fcb $40</td>
<td>fcb $40</td>
<td>CCR</td>
</tr>
<tr>
<td>fdb 0,0,0</td>
<td>fdb 0,0,0</td>
<td>fdb 0,0,0</td>
<td>DXY</td>
</tr>
<tr>
<td>fdb ProgA</td>
<td>fdb ProgA</td>
<td>fdb ProgB</td>
<td>PC</td>
</tr>
</tbody>
</table>

**Preemptive Thread Scheduler in C**

```c
void main(void){ DDRC=0xFF; RunPt=&sys[0]; /* Specify first thread */ asm(" sei"); TOC3vector=&ThreadSwitch; TFLG1 = 0x20; /* Clear OC3F */ TMSK1 = 0x20; /* Arm TDC3 */ TDC3=TCNT+20000; PORTB=RunPt->Id;
asm(" ldx _RunPt\n" " lds 2,x\n" " cli\n" " rti"); /* Launch First Thread */
}
```

**Preemptive Thread Scheduler in C (cont)**

```c
void ThreadSwitch(){
asm(" ldx _RunPt\n"
" sts 2,x");
RunPt=RunPt->Next;
PORTB=RunPt->Id; /* PortB=active thread */
asm(" ldx _RunPt\n"
" lds 2,x");
TDC3=TCNT+20000; /* Thread runs for 10 ms */
TFLG1=0x20; } /* ack by clearing TDC3F */
```

**Preemptive Thread Scheduler in Assembly**

Next equ 0 pointer to next TCB
SP equ 2 Stack pointer for this thread
Id equ 4 Used to visualize thread running
RunPt rm 2 pointer to thread running
Main ldaa #$FF
staa DDRC PortC displays program executing
ldx #TCB1 First thread to run
jmp Start
* Suspend thread which is currently running
OC5Han ldx RunPt
sts SP,x save Stack Pointer in TCB
Preemptive Thread Scheduler in Assembly (cont)

* launch next thread
  ldx Next,x
Start  stx RunPt
    ldaa Id,x
    staa PORTB visualizes running thread
  lds SP,x  set SP for this new thread
  ldd TOC5
  addd #20000  interrupts every 10 ms
  std TOC5
  ldaa #$08  ($20 on the 6812)
  staa TFLG1  acknowledge OC5
  rti

Other Scheduling Algorithms

* A non-preemptive (cooperative) scheduler trusts each thread to voluntarily release control on a periodic basis.
* Not appropriate for real-time systems.
* A priority scheduler assigns a priority to each thread.
* A thread is scheduled only if no higher priority thread is ready.
* Priority reduces latency for important tasks.
* In a busy system, low-priority threads may starve.

Profile of Three Threads

(See Figure 5.7)

Dynamic Allocation of Threads

void create(void (*program)(void), int TheId){
  TCBPtr NewPt;  // pointer to new TCB
  NewPt=(TCBPtr)malloc(sizeof(TCBType));
  if(NewPt==0) return;
  NewPt->SP=&(NewPt->CCR-1);  /* SP when not running */
  NewPt->Id=TheId;  /* used to visualize */
  NewPt->CCR=0x40;  /* Initial CCR, I=0 */
  NewPt->RegB=0;  /* Initial RegB */
  NewPt->RegA=0;  /* Initial RegA */
  NewPt->RegX=0;  /* Initial RegX */
  NewPt->RegY=0;  /* Initial RegY */
  NewPt->PC=program;  /* Initial PC */
  if(RunPt){
    NewPt->Next=RunPt->Next;
    RunPt->Next=NewPt;  /* will run Next */
  } else
    RunPt=NewPt;  /* first & only thread */