EE 319K
Introduction to Embedded Systems

Lecture 4: Board, Control structures, Modular Programming, Subroutines, Parameter passing

Announcements

- Lab partners
  - Select partner for labs 3-9
  - Email instructor if you don't have a partner
- Boards
  - Get one used or new board per team
    - New boards can be picked up in 2nd-floor checkout after payment has cleared
  - Bring board to assigned lab slot next week
    - TAs will check your board and give tutorial
  - Do not power up boards before you have received instructions by the TAs
    - Use at your own risk

Agenda

- Recap
  - Debugging
  - Arithmetic
    - Addition/subtraction, condition codes
    - Overflow and floor/ceiling, conditionals
- Outline
  - Board intro
  - Design process revisited
  - Control structures
    - If-then, loops
  - Modular programming
    - Subroutines, simple parameter passing

TExas in Real Mode

We will run **`Simple`** in both Simulation mode and Real Mode. Note:

- Use the cutout sheet between the 9S12 board and the breadboard to help identify pins
- If you do not know the COM port then enter 0
- Make all connections/disconnections with power supply off
- Place 9S12 board in “LOAD” mode when loading and debugging code
- To use embedded system mode (8 MHz) switch the board to RUN mode and you may disconnect the RS232 cable interface.
- Is your PP7 light flashing three times slower in RUN mode

Problem Solving

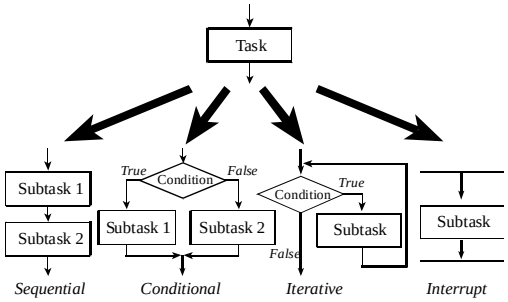
When we solve problems on the computer, we need to answer these questions:

- What does being in a state mean?
  - List state parameters
- What is the starting state of the system?
  - Define the initial state
- What information do we need to collect?
  - List the input data
- What information do we need to generate?
  - List the output data
- How do we move from one state to another?
  - Actions we could do
- What is the desired ending state?
  - Define the ultimate goal

Successive Refinement

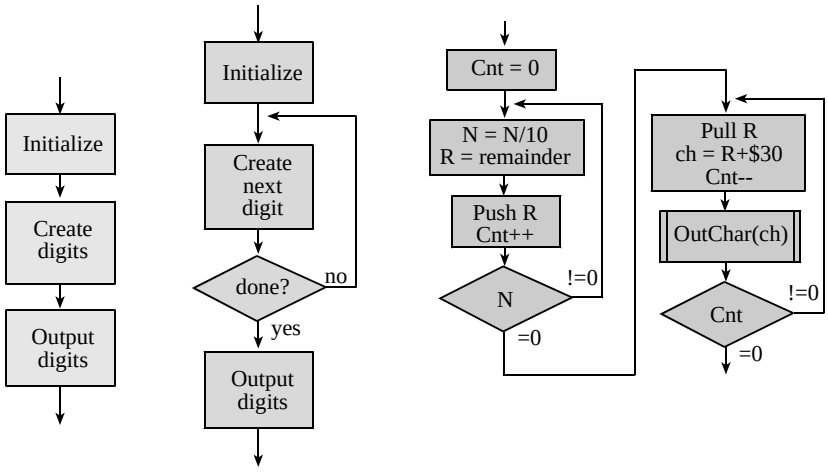
- Start with a task and decompose the task into a set of simpler subtasks
- Subtasks are decomposed into even simpler sub-subtasks.
- Each subtask is simpler than the task itself.
- Make design decisions
- Subtask is so simple, it can be converted to software code.

Decomposition



“do A then do B”	→ sequential
“do A and B in either order”	→ sequential
“if A, then do B”	→ conditional
“for each A, do B”	→ iterative
“do A until B”	→ iterative
“repeat A over & over forever”	→ iterative (condition always true)
“on external event do B”	→ interrupt
“every t msec do B”	→ interrupt

Successive Refinement: Example

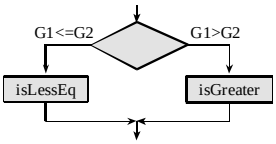


Ramesh Yerraballi

4-8

Given a number N display it in decimal form. Eg., if N=4521; we want to see the output show 4521. The constraint is that OutChar(ch) can only print an ASCII character.

if-then-else

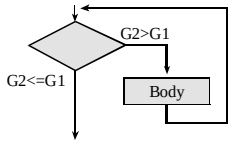



```
      ldaa G1                          if(G1>G2){
      cmpa G2
      bhi  high  ; branch if G1>G2
low   jsr  isLessEq   ; G1<=G2           isGreater();
      bra  next                        }
                                       else{
high  jsr  isGreater  ; G1>G2            isLessEq();
next                                   }
```

Two alternative implementations

```
      ldaa G1                          if(G1>G2){
      cmpa G2
      bls  low  ; branch if G1≤G2
high  jsr  isGreater  ; G1>G2            isGreater();
      bra  next                        }
                                       else{
low   jsr  isLessEq   ; G1<=G2           isLessEq();
next                                   }
```

while loop

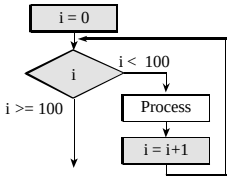



```
loop ldaa G2
     cmpa G1
     bls  next  ;stop if G2≤G1
     jsr  Body  ;body of loop
     bra  loop
next
```

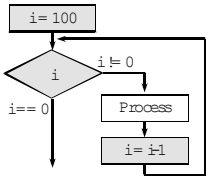
```
while(G2 > G1){
  Body();
}
```

for loop

```
for(i=0; i<100; i++){
  Process();
}
```




```
for(i=100; i!=0; i--){
  Process();
}
```




```
      ldab  #0        ; i=0
loop  cmpb  #100
      bhs   done
      jsr   Process
      incb            ; i=i+1
      bra   loop
done
```

```
for(i=0; i<100; i++){
  Process();
}
```

```
      ldab  #100       ; i=100
L1    jsr   Process
      dbne B,L1        ;i=i-1
done
```

```
for(i=100; i!=0; i--){
  Process();
}
```

*The **dbne** instruction optimizes this for-loop implementation.*

Modular Design

- Goal
  - Clarity
  - Create a complex system from simple parts
- Definition of modularity
  - Maximize number of modules
  - Minimize bandwidth between them
- Entry point (where to start)
  - The label of the first instruction of the subroutine
- Exit point (where to end)
  - The **`rts`** instruction
  - Good practice, one **`rts`** as the last line

Modular Design

- Public (shared, called by other modules)
  - Add underline in the name, module name before
- Private (not shared, called only within this module)
  - No underline in the name
  - Helper functions
- Coupling (amount of interaction between modules)
  - Data passed from one to another (bandwidth)
  - Synchronization between modules

Subroutines and the Stack

classical definition of the stack

- push saves data on the top of the stack,
- pull removes data from the top of the stack
- stack implements last in first out (LIFO) behavior
- stack pointer (SP) points to top element

many uses of the stack

- temporary calculations
- subroutine (function) return addresses
- subroutine (function) parameters
- local variables

- **psha** push Register A on the stack
- **pshb** push Register B on the stack
- **pshx** push Register X on the stack
- **pshy** push Register Y on the stack
- **des** S=S-1(reserve space)
- **pula** pull from stack into A
- **pulb** pull from stack into B
- **pulx** pull from stack into X
- **puly** pull from stack into Y
- **ins** S=S+1 (discard top of stack)

Registers to pass parameters

High level program	Subroutine
1) Sets Registers to contain inputs	
2) Calls subroutine	
	3) Sees the inputs in registers
	4) Performs the action of the subroutine
	5) Places the outputs in registers
6) Registers contain outputs	