AVR Programming II

ISA

- Addressing
- Instructions

Addressing

- Direct access
  - Register direct
  - I/O direct
  - Data direct
- Indirect data space access
  - Data indirect
  - Data indirect with displacement
  - Data indirect with pre-decrement
  - Data indirect with post-increment
- Program access
  - Direct program access
  - Indirect program access
  - Relative program access

Instructions

- Arithmetic and logic instructions
  - **ADD** (SUB), **SUBI** (no ADI), **ADIW**
  - AND, ANDI
  - **COM**, NEG
  - **SBR**, CBR
  - INC, DEC
  - **CLR**, SER
  - **MUL**
  - Flags in SREG (eg. what is V for ADD?)

Instructions

- Branch instructions
  - JMP, **RJMP**, **IJMP**
  - **CALL**, RCALL, ICALL
  - **RET**, **RETI**
  - CP, **CPI**
  - BRXX
  - CPSE, SBXX
  - Range of branch?

Instructions

- Data transfer instructions
  - **LD** (ST), **LDI** (no STI), **LDD** (STD)
  - MOV, **MOVW**
  - **IN**, OUT
  - PUSH, POP
  - Target address?

Instructions

- Bit and bit-test instructions
  - **SBI**, CBI (for I/O registers)
  - **BSET**, BCLR (for SREG)
  - **SEX**, CLX (for SREG's X flag)
  - Target registers?
- MCU control instruction
  - NOP
  - **SLEEP**

Example: Case 10

- for (i=0;i<k;i++) j=j+i;

Assume r12:13 for j, r14 for i, r16 for k.

```
clr r12
clr r13
clr r14
clr r15
ForBegin:
cp r14, r16
brlo ForLoop
rjmp ForEnd
  ForLoop:
  add r12, r14
  adc r13, r15
  inc r14
  rjmp ForBegin
ForEnd:
```

Example: Case 10

- y=foo(x=20);

```
Assume r16 is for x
Assume r2:3 is for y
Assume r0:1 is for return
  ldi r16, 20
  call FOO
  mov r2, r0
  mov r3, r1
```

```
int foo(int k) {
  int i,j;
  j=0;
  for (i=0;i<k;i++)
    j=j+i;
  return j;
}
```

```
FOO:
push r12
push r13
push r14
push r15
push r16
clr r12
clr r13
clr r14
clr r15
ForBegin:
cp r14, r16
brlo ForLoop
rjmp ForEnd
```

```
 ForLoop:
 add r12, r14
 adc r13, r15
 inc r14
 rjmp ForBegin
ForEnd:
mov r0, r12
mov r1, r13
pop r16
pop r15
pop r14
pop r13
pop r12
ret
```

Example: Case 11

```
  ... ;
  c2: 0e 94 67 00    jmp  0xce      ; jump to <main> at 0xce.
  ... ;
000000ce <main>:                    ; the stack pointer of <main> is at 0x10FF.
  ce: c7 ef          ldi  r28, 0xF7 ; move up the stack pointer to 0x10F7, because b[8] is at 0x10F8.
  d0: d0 e1          ldi  r29, 0x10 ;
  d2: de bf          out  0x3e, r29 ; store the new stack pointer.
  d4: cd bf          out  0x3d, r28 ;
  d6: 80 e0          ldi  r24, 0x00 ; a[] is at 0x0100
  d8: 91 e0          ldi  r25, 0x01 ; the address of a[] is loaded in r24:25
  da: bc 01          movw r22, r24  ; r22:23 are used as the second parameter of strcpy (a[])
  dc: ce 01          movw r24, r28  ; r24:25 are used as the first parameter of strcpy (b[])
  de: 01 96          adiw r24, 0x01 ;
  e0: 0e 94 76 00    call 0xec      ; call <strcpy> at 0xec.
  e4: 80 e0          ldi  r24, 0x00 ;
  e6: 90 e0          ldi  r25, 0x00 ;
  e8: 0c 94 7d 00    jmp  0xfa      ; jump to <_exit> at 0xfa.
000000ec <strcpy>:                  ; the stack pointer of <strcpy> is at 0x10F5.
  ec: fb 01          movw r30, r22  ; The second parameter (a[])is moved to Z.
  ee: dc 01          movw r26, r24  ; The first parameter (b[]) is moved to X.
  f0: 01 90          ld   r0, Z+    ; load a byte (char) at Z to r0
  f2: 0d 92          st   X+, r0    ; store r0 to X
  f4: 00 20          and  r0, r0    ;
  f6: e1 f7          brne .-8       ; loop until r0 is 0x00.
  f8: 08 95          ret            ;
000000fa <_exit>:
  fa: ff cf          rjmp .-2       ; jump to itself
```

Example: Case 12

```
__vectors:
vector     __vector_1
vector     __vector_2
......
vector     __vector_35



__init:
clr __zero_reg__
out AVR_STATUS_ADDR, __zero_reg__
ldi r28,lo8(__stack)
ldi r29,hi8(__stack)
out AVR_STACK_POINTER_HI_ADDR, r29


__do_copy_data:
ldi r17, hi8(__data_end)
ldi r26, lo8(__data_start)
ldi r27, hi8(__data_start)
ldi r30, lo8(__data_load_start)
ldi r31, hi8(__data_load_start)
ldi r16, hh8(__data_load_start)
out AVR_RAMPZ_ADDR, r16
rjmp .L__do_copy_data_start
.L__do_copy_data_loop:
elpm r0, Z+
st  X+, r0
.L__do_copy_data_start:
cpi r26, lo8(__data_end)
cpc r27, r17
brne .L__do_copy_data_loop

XJMP main
```

```
__vectors:
jmp L0188  ; 0x0000
jmp L01A5  ; 0x0002
........
jmp L01A5  ; 0x0044

.DW 0xBAAB ; 0x0046
......
.DW 0x0081 ; 0x0187

__init:
clr r1     ; 0x0188
out 0x3F, r1     ; 0x0189
ldi r28, 0xF0    ; 0x018A
ldi r29, 0x10    ; 0x018B
out 0x3E, r29    ; 0x018C
out 0x3D, r28    ; 0x018D

__do_copy_data:
ldi r17, 0x1     ; 0x018E
ldi r26, 0x0     ; 0x018F
ldi r27, 0x1     ; 0x0190
ldi r30, low(L11C0*2)  ; 0x0191
ldi r31, high(L11C0*2) ; 0x0192
ldi r16, 0x0     ; 0x0193
out 0x3B, r16    ; 0x0194
rjmp L0198 ; 0x0195
L0196:
elpm r0, Z+      ; 0x0196
st  X+, r0 ; 0x0197
L0198:
cpi r26, 0xA     ; 0x0198
cpc r27, r17     ; 0x0199
brbc 1, L0196    ; 0x019A

jmp L0BF6  ; 0x01A3, to L0BF6
```