Recursive Algorithm in AVR

Tim Dosen POK

Outline

- Recursive Addition
- Factorial
- Fibonacci

RecAdd: recursive algorithm (1)

```
int Add(int m, int n) {
  if (n == 0) return m;
  else {
    int y = Add(m, n-1) + 1;
    return y;
  }
}
```

RecAdd: recursive algorithm (2)

```
        .def Temp=r16
        .def Num1=r5
        .def Num2=r6
        .def Rslt=r7

        Reset:
0000    ldi     Temp, low(RAMEND)
0001    out     SPL,Temp
0002    ldi     Temp, high(RAMEND)
0003    out S   PH,Temp

0004    ldi     Temp, 7             ; first number
0005    mov     Num1, Temp
0006    ldi     Temp, 18            ; second number
0007    mov     Num2, Temp
0008    rcall   recadd
        forever:
0009    rjmp    forever
```

SRAM	Address
FF	0x0000
…	
FF	
FF	
FF	
FF	
FF	
FF	
FF	
FF	0x025F

RecAdd: recursive algorithm (2)

```
        .def Temp=r16
        .def Num1=r5
        .def Num2=r6
        .def Rslt=r7

        Reset:
0000    ldi      Temp, low(RAMEND)
0001    out      SPL,Temp
0002    ldi      Temp, high(RAMEND)
0003    out S    PH,Temp

0004    ldi      Temp, 7            ; first number
0005    mov      Num1, Temp
0006    ldi      Temp, 18           ; second number
0007    mov      Num2, Temp
0008    rcall    recadd
        forever:
0009    rjmp     forever
```

push

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
FF	
FF	
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
int Add ( int m, int n) {
/* base case */
   if ( n == 0 ) return m;


/* recursive case */
      else {
            int y = Add ( m, n-1 ) + 1;
            return y;
       }
}
```

```
recadd:
        tst Num2
        brne notzero
        mov Rslt, Num1
        ret

notzero:
        dec Num2
        rcall recadd
        inc Rslt
        ret
```

tst

TST – Test for Zero or Minus

Description:

Tests if a register is zero or negative. Performs a logical AND between a register and itself. The register will remain unchanged.

Operation:

(i) Rd ← Rd • Rd

	Syntax:	**Operands:**	**Program Counter:**
(i)	TST Rd	$0 \le d \le 31$	PC ← PC + 1

16-bit Opcode: (see AND Rd, Rd)

0010	00dd	dddd	dddd

brne

BRNE – Branch if Not Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was not equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction ($PC - 63 \leq destination \leq PC + 64$). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 1,k).

Operation:

(i) If $Rd \neq Rr$ ($Z = 0$) then $PC \leftarrow PC + k + 1$, else $PC \leftarrow PC + 1$

	Syntax:	Operands:	Program Counter:
(i)	BRNE k	$-64 \leq k \leq +63$	$PC \leftarrow PC + k + 1$ $PC \leftarrow PC + 1$, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

RecAdd: recursive algorithm (3)

	recadd:	
000A		**tst Num2**
000B		**brne notzero**
000C		**mov Rslt, Num1**
000D		**ret**
	notzero:	
000E		**dec Num2**
000F		**rcall recadd**
0010		**inc Rslt**
0011		**ret**

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
FF	
FF	
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
00	
10	0x025D
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
     recadd:
000A        tst Num2
000B        brne notzero
000C        mov Rslt, Num1
000D        ret

     notzero:
000E        dec Num2
000F        rcall recadd
0010        inc Rslt
0011        ret
```

SRAM	Address
FF	0x0000
...	
FF	
FF	
00	
10	0x025B
00	
10	0x025D
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

… and so on, until Num2 is zero

SRAM	Address
FF	0x0000
…	
00	
10	0x0259
00	
10	0x025B
00	
10	0x025D
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

… and so on, until Num2 is zero

SRAM	Address
FF	0x0000
…	
00	0x0228
10	0x0229
00	
10	
00	
…	
…	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

SRAM	Address
FF	0x0000
...	
00	0x0228
10	0x0229
00	
10	
00	
...	
...	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

pop!

SRAM	Address
FF	0x0000
...	
00	0x0228
10	0x0229
00	
10	
00	
...	
...	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

pop!

SRAM	Address
FF	0x0000
...	
FF	0x0228
FF	0x0229
00	
10	
00	
...	
...	
09	0x025F

RecAdd: recursive algorithm (3)

```
       recadd:
000A               tst Num2
000B               brne notzero
000C               mov Rslt, Num1
000D               ret

       notzero:
000E               dec Num2
000F               rcall recadd
0010               inc Rslt
0011               ret
```

pop!

SRAM	Address
FF	0x0000
...	
FF	0x0228
FF	0x0229
00	
10	
00	
...	
...	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

pop!

SRAM	Address
FF	0x0000
…	
FF	0x0228
FF	0x0229
FF	
FF	
00	
…	
…	
09	0x025F

… and so on …

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

SRAM	Address
FF	0x0000
…	
FF	
FF	
FF	
FF	
FF	
FF	
00	
09	0x025F

RecAdd: recursive algorithm (3)

```
        recadd:
000A            tst Num2
000B            brne notzero
000C            mov Rslt, Num1
000D            ret

        notzero:
000E            dec Num2
000F            rcall recadd
0010            inc Rslt
0011            ret
```

	SRAM	Address
	FF	0x0000
	...	
	FF	
	FF	
	FF	
	FF	
	FF	
	FF	
	00	
pop!	09	0x025F

RecAdd: recursive algorithm (2)

```
     .def Temp=r16
     .def Num1=r5
     .def Num2=r6
     .def Rslt=r7

     Reset:
0000  ldi     Temp, low(RAMEND)
0001  out     SPL,Temp
0002  ldi     Temp, high(RAMEND)
0003  out S   PH,Temp

0004  ldi     Temp, 7          ; first number
0005  mov     Num1, Temp
0006  ldi     Temp, 18         ; second number
0007  mov     Num2, Temp
0008  rcall   recadd
     forever:
0009  rjmp    forever
```

SRAM	Address
FF	0x0000
…	
FF	
FF	
FF	
FF	
FF	
FF	
FF	
FF	0x025F

Factorial: recursive algorithm (1)

```
int factorial(int n) {
  if (n == 0) return 1;
  else return (n * factorial(n - 1));
}
```

Factorial: recursive algorithm (1)

```
        .include "m8515def.inc"
        .def      temp = r18
        .def      tempin = r19        ; Define temporary variable
        .def      tempout = r20

        start:
0000    ldi       temp,low(RAMEND)
0001    out       SPL,temp
0002    ldi       temp,high(RAMEND)
0003    out       SPH,temp

0004    ldi       r16, 5
0005    mov       tempin, r16
0006    rcall     fact
        forever:
0007    rjmp      forever
```

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
FF	
FF	
FF	
FF	0x025F

Factorial: recursive algorithm (1)

```
        .include "m8515def.inc"
        .def    temp = r18
        .def    tempin = r19     ; Define temporary variable
        .def    tempout = r20

        start:
0000    ldi     temp,low(RAMEND)
0001    out     SPL,temp
0002    ldi     temp,high(RAMEND)
0003    out     SPH,temp

0004    ldi     r16, 5
0005    mov     tempin, r16
0006    rcall   fact
        forever:
0007    rjmp    forever
```

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
FF	
FF	
00	
07	0x025F

push!

Factorial: recursive algorithm (2)

```
      fact:
000A   push temp
000B   push tempin
      basecase:
000C   tst tempin
000D   breq endcase
      reccase:
000E   mov temp, tempin
000F   dec tempin
0010   rcall fact
0011   mul temp, tempout
0012   mov tempout, r0
0013   rjmp outrec
      endcase:
0014   ldi tempout, 1
      outrec:
0015   pop tempin
0016   pop temp
0017   ret
```

	SRAM	Address
	FF	0x0000
	...	
	FF	
	FF	
	FF	
	FF	
	FF	
push!	02	
	00	
	07	0x025F

Factorial: recursive algorithm (2)

```
      fact:
000A    push temp
000B    push tempin
      basecase:
000C    tst tempin
000D    breq endcase
      reccase:
000E    mov temp, tempin
000F    dec tempin
0010    rcall fact
0011    mul temp, tempout
0012    mov tempout, r0
0013    rjmp outrec
      endcase:
0014    ldi tempout, 1
      outrec:
0015    pop tempin
0016    pop temp
0017    ret
```

SRAM	Address
FF	0x0000
…	
FF	
FF	
FF	
FF	
05	
02	
00	
07	0x025F

push!

Factorial: recursive algorithm (2)

```
        fact:
000A      push temp
000B      push tempin
        basecase:
000C      tst tempin
000D      breq endcase
        reccase:
000E      mov temp, tempin
000F      dec tempin
0010      rcall fact
0011      mul temp, tempout
0012      mov tempout, r0
0013      rjmp outrec
        endcase:
0014      ldi tempout, 1
        outrec:
0015      pop tempin
0016      pop temp
0017      ret
```

	SRAM	Address
	FF	0x0000
	…	
	FF	
	FF	
	00	
push!	11	
	05	
	02	
	00	
	07	0x025F

Factorial: recursive algorithm (2)

```
      fact:
000A    push temp
000B    push tempin
      basecase:
000C    tst tempin
000D    breq endcase
      reccase:
000E    mov temp, tempin
000F    dec tempin
0010    rcall fact
0011    mul temp, tempout
0012    mov tempout, r0
0013    rjmp outrec
      endcase:
0014    ldi tempout, 1
      outrec:
0015    pop tempin
0016    pop temp
0017    ret
```

	SRAM	Address
	FF	0x0000
	…	
	FF	
	..	
	FF	
	FF	
	00	
push!	11	
	…	
	07	0x025F

after some more iterations…

Factorial: recursive algorithm (2)

```
      fact:
000A   push temp
000B   push tempin
      basecase:
000C   tst tempin
000D   breq endcase
      reccase:
000E   mov temp, tempin
000F   dec tempin
0010   rcall fact
0011   mul temp, tempout
0012   mov tempout, r0
0013   rjmp outrec
      endcase:
0014   ldi tempout, 1
      outrec:
0015   pop tempin
0016   pop temp
0017   ret
```

SRAM	Address
FF	0x0000
…	
FF	
..	
00	
00	
00	
11	
…	
07	0x025F

push!

Factorial: recursive algorithm (2)

```
      fact:
000A    push temp
000B    push tempin
      basecase:
000C    tst tempin
000D    breq endcase
      reccase:
000E    mov temp, tempin
000F    dec tempin
0010    rcall fact
0011    mul temp, tempout
0012    mov tempout, r0
0013    rjmp outrec
      endcase:
0014    ldi tempout, 1
      outrec:
0015    pop tempin
0016    pop temp
0017    ret
```

pop!

SRAM	Address
FF	0x0000
…	
FF	
..	
00	
00	
00	
11	
…	
07	0x025F

Factorial: recursive algorithm (2)

```
      fact:
000A   push temp
000B   push tempin
      basecase:
000C   tst tempin
000D   breq endcase
      reccase:
000E   mov temp, tempin
000F   dec tempin
0010   rcall fact
0011   mul temp, tempout
0012   mov tempout, r0
0013   rjmp outrec
      endcase:
0014   ldi tempout, 1
      outrec:
0015   pop tempin
0016   pop temp
0017   ret
```

pop!

SRAM	Address
FF	0x0000
…	
FF	
..	
00	
00	
00	
11	
…	
07	0x025F

Factorial: recursive algorithm (2)

```
      fact:
000A    push temp
000B    push tempin
      basecase:
000C    tst tempin
000D    breq endcase
      reccase:
000E    mov temp, tempin
000F    dec tempin
0010    rcall fact
0011    mul temp, tempout
0012    mov tempout, r0
0013    rjmp outrec
      endcase:
0014    ldi tempout, 1
      outrec:
0015    pop tempin
0016    pop temp
0017    ret
```

pop!

SRAM	Address
FF	0x0000
...	
FF	
..	
00	
00	
00	
11	
...	
07	0x025F

Factorial: recursive algorithm (2)

```
        fact:
000A    push temp
000B    push tempin
        basecase:
000C    tst tempin
000D    breq endcase
        reccase:
000E    mov temp, tempin
000F    dec tempin
0010    rcall fact
0011    mul temp, tempout
0012    mov tempout, r0
0013    rjmp outrec
        endcase:
0014    ldi tempout, 1
        outrec:
0015    pop tempin
0016    pop temp
0017    ret
```

after some more iterations…

pop!

SRAM	Address
FF	0x0000
…	
…	
…	
…	
…	
05	
02	
00	
07	0x025F

Factorial: recursive algorithm (1)

```
       .include "m8515def.inc"
       .def     temp = r18
       .def     tempin = r19      ; Define temporary variable
       .def     tempout = r20

       start:
0000   ldi      temp,low(RAMEND)
0001   out      SPL,temp
0002   ldi      temp,high(RAMEND)
0003   out      SPH,temp

0004   ldi      r16, 5
0005   mov      tempin, r16
0006   rcall    fact
       forever:
0007   rjmp     forever
```

SRAM	Address
FF	0x0000
…	
…	
…	
…	
…	
05	
02	
00	
07	0x025F

Fibonacci: recursive algorithm (1)

```
int fib(int n) {
  if (n <= 1)  return n;
  else return (fib(n-1) + fib(n-2));
}
```

Fibonacci: recursive algorithm (1)

```
        .include "m8515def.inc"

        .def rone  = r1
        .def input = r16
        .def rslt  = r17
        .def less1 = r4
        .def less2 = r18
        .def temp  = r19

        START:
0000            ldi     temp,low(RAMEND) ; Set stack pointer to -
0001            out     SPL,temp   ; -- last internal RAM location
0002            ldi     temp,high(RAMEND)
0003            out     SPH,temp

0004            ldi temp, 1
0005            mov rone, temp

0006            ldi input, 9
0007            rcall fib
        forever:
0008            rjmp forever
```

Push!

SRAM	Address
FF	0x0000
...	
FF	
FF	
FF	
FF	
FF	
FF	
00	
08	0x025F

Fibonacci: recursive algorithm (2)

```
        fib:
0009        push input          ; Push into the stack
000A        push less1
000B        push less2
000C        push temp

000D        tst input
000E        breq zero
000F        cp input, rone
0010        brbs 1, one

0011        mov less1, input
0012        dec less1
0013        mov less2, input
0014        subi less2, 2

0015        mov input, less1
0016        rcall fib
0017        mov temp, rslt

0018        mov input, less2
0019        rcall fib
001A        add rslt, temp
001B        rjmp done
```

SRAM	Address
FF	0x0000
…	
FF	
FF	
01	
00	
00	
09	
00	
08	0x025F

BRBS

BRBS – Branch if Bit in SREG is Set

Description:

Conditional relative branch. Tests a single bit in SREG and branches relatively to PC if the bit is set. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form.

Operation:

(i) If SREG(s) = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRBS s,k	$0 \le s \le 7$, $-64 \le k \le +63$	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	ksss

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0

CP

CP – Compare

Description:

This instruction performs a compare between two registers Rd and Rr. None of the registers are changed. All conditional branches can be used after this instruction.

	Operation:
(i)	Rd - Rr

	Syntax:	**Operands:**	**Program Counter:**
(i)	CP Rd,Rr	$0 \leq d \leq 31$, $0 \leq r \leq 31$	$PC \leftarrow PC + 1$

16-bit Opcode:

0001	01rd	dddd	rrrr

Fibonacci: recursive algorithm (2)

```
fib:
0009    push input
000A    push less1
000B    push less2
000C    push temp

000D    tst input
000E    breq zero
000F    cp input, rone
0010    brbs 1, one

0011    mov less1, input
0012    dec less1
0013    mov less2, input
0014    subi less2, 2

0015    mov input, less1
0016    rcall fib          ; We recurse with f(input –1) (which is
0017    mov temp, rslt     now f(8))

0018    mov input, less2   ; Calculate f(input-2),then recurs
0019    rcall fib
001A    add rslt, temp     ; Recursion will happen until resulting
001B    rjmp done          value of 1 or 0. During recursion these
                           1's and 0's are added to the rslt.
                           e.g. f(3) = f(2) + f(1) = f(1) + f(0) +f (1)
```

SRAM	Address
FF	0x0000
…	
00	
17	
01	
00	
00	
09	
00	
08	0x025F

Current Stack frame

Fibonacci: recursive algorithm (2)

```
        zero:
001C                ldi rslt, 0      ; Case if f(0)
001D                rjmp done
        one:
                    ldi rslt, 1      ;Case if f(1)
001E    done:
001F                pop temp
0020                pop less2
0021                pop less1
0022                pop input
                    ret
```

SRAM	Address
FF	0x0000
FF	
..	
XX	
XX	
XX	
XX	
XX	
..	
08	0x025F

Pop!

; pop our stack to take our next value, returns to the previous instruction (from stack) then recurs again.

;** Noted that XX is some value

Fibonacci: recursive algorithm (2)

```
fib:
        push input
        push less1
        push less2
        push temp

        tst input
        breq zero
        cp input, rone
        brbs 1, one

        mov less1, input
        dec less1
        mov less2, input
        subi less2, 2

        mov input, less1
        rcall fib
        mov temp, rslt

        mov input, less2
        rcall fib
        add rslt, temp
        rjmp done

zero:
        ldi rslt, 0
        rjmp done
one:
        ldi rslt, 1
done:
        pop temp
        pop less2
        pop less1
        pop input
        ret
```

```
int fib(int n) {
  if (n <= 1)
        return n;
  else
        return (fib(n-1) + fib(n-2));
}
```