ASxxxx Tool을 이용한 NH800 Cross-Assembler 구현*

김민성, 나여울, 김선욱
고려대학교 전기전자전파공학과
e-mail: {kissofgod,rapidsna,seon}@korea.ac.kr

Implementation of NH800 Cross-Assembler Using ASxxxx Tool

Min Seong Kim, Yeoul Na, Seon Wook Kim
School of Electrical Engineering, Korea University

요 약

스마트 기기의 급격한 발전, 의료분야와 IT 기술의 융합, 소비자들의 높아지는 요구로 인해 8 비트 이하 마이크로 컨트롤러의 활용도가 높아질 전망이다. 이 논문에서는 EISC구조의 8 비트 마이크로 컨트롤러인 NH800의 ISA를 소개하고, 마이크로 컨트롤러를 위한 오픈소스 컴파일러인 SDCC에서 사용하는 ASxxxx Tool을 이용한 NH800 어셈블러 포팅을 제시한다.

1. 서론

오늘날의 휴대기기는 점점 얇아지고 가벼워지고 있다. 이러한 휴대성에 대한 요구는 극대화 되어 portable 기기에서 wearable 기기로의 진화가 이루어지고 있다. 전자 기기가 인체에 부착되어 지속적인 외부의 전력공급이 없이도 반영구적인 수명을 갖기 위해서는 저전력을 넘어선 uW 단위의 미세전력 시스템이 요구되며, "저탄소 녹색성장"을 향한 세계적인 추세 역시 초저전력으로의 확실한 방향성을 보여준다. 이러한 요구에 발맞춘 초저전력 시스템 설계에는 32 비트 이상의 고성능 컨트롤러보다는, 작은 코드사이즈, 높은 코드 밀도, 간단한 명령어 집합과 하드웨어, 저전력, 가격경쟁력 특성을 가지고 있는, 8 비트 이하의 로우엔드 마이크로 컨트롤러가 더욱 효과적으로 활용될 전망이다. 현재 8 비트 컨트롤러의 경우 전체 MCU의 50%이상을 차지하고 있으나 [1], 대부분의 마이크로 컨트롤러들은 INTEL의 8051, ATMEL의 AVR 등의 외국산 제품이 주를 이루고 있어 해마다 상당액의 로열티를 지불해야 하며, 점차 다양해지는 임베디드 시스템의 요구에 맞추기 위해 시스템의 요구사항에 최적화된 고유한 ISA를 가진 마이크로 컨트롤러의 수요가 커질 전망이다 [2].

이 논문에서는 이처럼 새로운 ISA를 가진 마이크로 컨트롤러의 중요성이 대두되는 상황에서 향후 다양한 마이크로 컨트롤러의 어셈블러 구현에 대한 지침을 제시하기 위해, NH800 마이크로 컨트롤러를 타겟으로 한 ASxxxx 어셈블러 포팅방법을 제시한다. 이 논문에서 제시하는 NH800은 독자적인 ISA를 가지는 EISC 기반의 8 비트 마이크로 컨트롤러이며, ASxxxx Cross-Assembler는 마이크로 컨트롤러를 위한 오픈소스 컴파일러인 Small Device C Compiler (SDCC)에서 사용하는 어셈블러이다.

이 논문의 구성은 다음과 같다. 2장에서는 ASxxxx Cross-Assembler 에 대해 설명한다. 3장에서는 NH800의 ISA 특성을 설명한다. 4장에서는 NH800의 ISA 특성에 따라 ASxxxx Tool을 이용하여 어셈블러를 구현한 방법을 기술하고, 5장에서는 실제 구현된 모습을 생성된 Target Code와 함께 제시한다.

2. ASxxxx Cross-Assembler

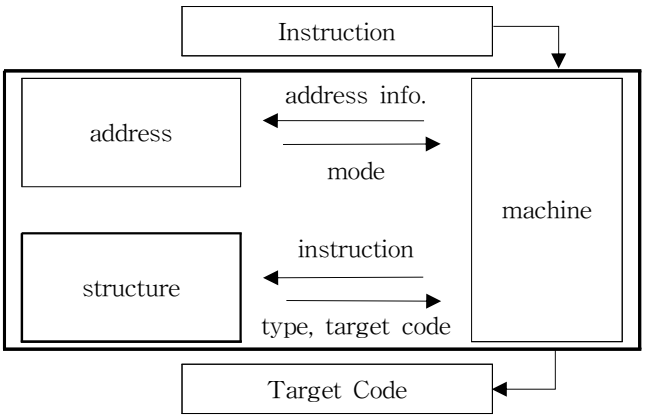
ASxxxx Cross-Assembler는 기본적으로 (그림 1)과 같은 구조를 가진다. Address의 Mode을 구분해 주는 address부분, 각각의 명령어에 대응되는 Type과 Opcode를 저장하고 있는 structure 부분, 그리고 앞선 부분의 데이터를 이용해 Target Code를 생성해주는 machine 부분이 있다.

(그림 2)는 (그림 1)의 address 코드의 일부분이다. 이 코드는 Operand의 Mode를 설정해준다. 한 예로 (그림 2)에서 보는 것과 같이 '#'으로 시작하는 Immediate Data일 경우, mode를 S_IMMED로 설정해준다.

(그림 3)은 (그림 1)의 structure 코드의 일부분이다. 이 코드는 명령어에 대해 대응하는 Type과 Target Code에 대한 정보를 담고 있다. (그림 3)에서 보는 것과 같이

이 논문은 2011 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No.2011-0020128).

'add'의 Type은 S_T6121이고 Target Code는 0xC0 이다.



(그림 1) ASxxxx Cross-Assembler 구조

```
int
addr(esp)
struct expr *esp;
{
      if ((c = getnb()) == '#') {
      expr(esp, 0);
      esp->e_mode = S_IMMED;
      } else
      if ((c == '@') || (c == LFIND)) {
      if ((indx = admode(R))) {/*      @R or (R)      */
      mode = S_IR;
      } else
      if ((indx = admode(RR))) {/*     @RR or (RR)     */
      mode = S_IRR;
```

(그림 2) Address Code

```
struct mne mne[] = {
{NULL,"add",S_T6121,0,0xC0},
{NULL,"adc",S_T6121,0,0xC8},
{NULL,"sub",S_T6121,0,0xD0},
{NULL,"sbc",S_T6121,0,0xD8},
```

(그림 3) Structure Code

(그림 4)는 (그림 1)의 machine 코드에서 structure, address 부분으로부터 Mode와 Type정보를 받아 실제 Target Code를 작성하는 부분을 보여준다. 한 예로 'push R' 의 명령어는 Single Operand인 S_SOP인 예이고, address을 통해 얻어진 mode가 S_R이다. 이때 structure에서 읽어온 Opcode가 'push R'에 해당하는 0x63인 경우로 해당하는 Opcode를 출력하게 된다.

이와 같이 ASxxxx Tool은 프로그래밍 과정에서 명령어 집합과 실행코드 형식을 손쉽게 변경 또는 개선하게 해준다[4]. 따라서 이 논문에서는 마이크로 컨트롤러에 대응되는 어셈블러를 제작하기 위해, 명령어 집합의 Type을 분류하고, ASxxxx Cross-Assembler의 구조에 맞는 코드를 작성한다.

```
VOID
machine(mp)
struct mne *mp;
{
      case S_SOP:
      t1 = addr(&e1);
      v1 = (int) e1.e_addr;
      if (t1 == S_R) {/* op    r   */
            if (op==0x63) {
                  outab(op);
                  outab(0x20+v1); //push R
            } else {
                  outab(op+1);
                  outab(0x28+v1); //pop R
```

(그림 4) Machine Code

3. NH800의 ISA

NH800은 EISC(Extendable Instruction Set Computer) 프로세서이다. EISC는 에이디칩스에서 개발한 임베디드 프로세서용 RISC 기반 명령어 집합으로, RISC의 간결성과 CISC의 확장성을 동시에 가진다. RISC 프로세서는 명령어 지정에 있어서 고정 필드 명령어 형태를 사용하여 명령어 해독이 간편한 반면, EISC 구조는 코드 밀도를 높이기 위해 각각의 명령어에 따라 필요한 필드를 더 효율적으로 배정하므로 명령어 필드 지정이 매우 복잡하다.

이 논문에서 사용한 NH800은 8 비트와 16비트 하이브리드 명령어를 취하고 있으며 8 비트 ALU, 16 비트 address / 8 비트 데이터버스를 가지고 있다. 8*8 비트의 GPR이외에 3*8 비트의 SPR(Special Purpose Register)을 가지고 있다.

EISC구조를 가지는 NH800의 명령어 집합의 대표적인 8 비트 길이의 명령어 집합을 <표 2>에, 16 비트 길이의 명령어 집합을 <표 3>에 나타내었다. Type 0, Type4, Type 5는 8 비트 길이의 명령어 집합을 가지고, 그 이외는 16 비트 길이의 명령어 집합을 가진다[3].

<표 2> 8 비트 길이의 명령어 집합

Type	Instruction	Opcode							
		7	6	5	4	3	2	1	0
Type 0	MOV dst, src	0	0	source			dest		
Type 4	INC reg	0	1	0	0	0	register		
	DEC reg	0	1	0	0	1	register		
Type 5	RET	0	1	0	1	1	0	1	0

<표 3> 16 비트 길이의 명령어 집합

Type	Instruction	Opcode							
		0x	0x	7	6	5	4	3	2-0
Type 6	MOV dst,(indx)	6	1	0	0	indx			dst
	MOV dst, (dindx)	6	1	1	0	dindx		0	dst
	MOVP dst, (dindx)	6	2	0	0	dindx		0	dst
	MOVP (dindx), src	6	2	0	1	dindx		0	src
	POP dst	6	3	0	0	1	0	1	reg
	ADD dst, src	6	4	0	0	src			dst
Type 7	MOV dst, (addr)	7	0	address					dst
	MOV (addr), src	7	1	address					src
Type 8	JMP addr	8	offset						
	CALL addr	9	offset						
Type 12	SUB dst, immd	D	0	immediate					dst

4. NH800의 ISA 특성에 따른 구현

4.1. Instruction Type 설계

NH800은 EISC Architecture로 고정된 길이의 명령어 집합을 사용하며 그 명령어 집합의 Type 구분이 용이한 구조이다. 8 비트의 경우 [7-4]비트를 보고 Type을 구분할 수 있고, 16 비트의 경우 [15-12]비트를 보고 Type을 구분할 수 있다. 하지만 같은 Type의 경우에도 Operand의 개수가 다르고, Opcode의 구성이 달라 Type 정보만으로는 명령어에 대응하는 Target Code를 생성할 수 없다. 따라서 명령어 Type을 기준으로 하되 명령어의 Operand 에 따른 다양한 Type을 추가적으로 구성하였다. 상위 4개의 비트를 보고 정확히 Target Code를 구성할 수 있는 Type 5와 Type 10은 그대로 사용하고 그 이외의 Type에 해당되는 명령어들은 다음과 같이 새로운 Type을 구성하였다. 그 중 Type 6-4의 명령어 집합은 두 개의 Type으로 나누어 구성하였고, 나머지 명령어 집합은 Increment, Shift, Single Operand, Double Operand의 Type으로 구성한다. 그 구성은 <표 4>에 나타내었다.

<표 4>. Instruction Type

S_type	Instruction
S_INC (Increment)	inc, dec, inx, dcx, jmp, call, jmpr, callr
S_SFT (Shift)	ror, cir, srl, rol
S_SOP (Single Operand)	push, pop, swi, leri
S_DOP (Double Operand)	mov, movp
S_T05	clc, stc, di, ei, ret, reti, halt
S_T641	add, adc, sub, sbc
S_T642	and, or, xor, cmp
S_T10	jnv, jv, jns, jp, js, jm, jnz, jne, jz, je, jnc, jhe, jc jl, jgt, jlt, jge, jle, jhi, jle

4.2. Instruction Operand Mode 설계

Type 별로 구분된 명령어 집합은 Type 정보만을 가지고 주어진 명령어를 Target Code로 표현해 내지 못한다. 명령어에 대응되는 Target Code를 얻기 위해서는 Operand에 대한 정보도 필요하다. 따라서 Operand의 address specifier의 정보를 찾아 address의 mode를 설정한다. NH800 마이크로 컨트롤러에서 쓰이는 Mode에는 Register. Double Register, Immediate, Index(Index-Register), Index Double Register, Address, Offset 가 있으며, 각각의 Operand의 Mode는 아래의 <표 5>과 같이 정의한다.

이렇게 정의된 Type과 그에 대응하는 Opcode, 그리고 Operand Mode정보를 이용하여 address, structure, machine을 작성하여 어셈블러를 구현한다.

<표 5> Instruction Operand Mode

Mode	Definition	Syntax
Register	S_R	r0
Double Register	S_RR	rr0
Immediate	S_IMMED	#0x12
Index(Index Register)	S_IR	(r0)
Index Double Register	S_IRR	(rr0)
Address	S_INDX	(0x12)
Offset	S_USER	0x12

5. 어셈블러의 실행과 검증

본 장에서는 구현된 NH800 Assembler를 통해 Sample Assembly Code를 입력으로 받아 그에 맞는 Target Code를 생성함으로써 어셈블러의 동작을 검증한다.

(그림 5)는 NH800 Assembler를 검증하기 위한 샘플코드이다. 8 비트 길이의 명령어 중 Operand가 없는 것, 하나인 것, 2 개인 것을 작성하였다. 16 비트 길이의 명령어에서도 8 비트와 마찬가지로 작성하고, Operand가 2 개인 것 중에서 index, double-index, immediate data offset value를 사용하는 것을 추가로 작성하였다. 또한 이 Sample Assembly Code에 대응되는 Opcode는 <표 2>와 <표 3>에 제시하였다.

(그림 5)에 작성된 샘플코드를 구현된 NH800 Assembler를 이용하여 Target Code(Hex)를 생성한 결과, (그림 6)과 같이 생성 되었다.

Sample Assembly Code

```
;8bit instruction
ret
mov r0, r1                ; mov dst, src
inc r2                    ; inc reg

;16bit instruction
pop r0                    ; pop reg
add r1,r2                 ; add dst, src
mov r3,(r4)               ; mov dst, index
mov r5,(rr0)              ; mov dst, double-index
mov r6,(0x01)             ; mov dst, addr
sub r0, #0x02             ; sub dst, immediate
jmp 0xa1                  ; jmp offset
```

(그림 5) Sample Assembly Code

```
                    8 ;8bit instruction
0000 5A             9 ret
0001 08             10 mov r0, r1
0002 42             11 inc r2
                    12
                    13 ;16bit instruction
0003 63 28          14 pop r0
0005 64 11          15 add r1,r2
0007 61 23          16 mov r3,(r4)
0009 61 85          17 mov r5,(rr0)
000B 70 0E          18 mov r6,(0x01)
000D D0 10          19 sub r0, #0x02
000F 80 A1          20 jmp 0xa1
```

(그림 6) Target Code

5. 결 론

이 논문에서는 ASxxxx Tool을 이용한 8 비트 마이크로 컨트롤러 NH800 어셈블러 구현 방법을 제시하였으며, NH800의 어셈블리 코드를 입력으로 받아들여 그에 대응하는 Target Code가 생성됨을 보였다. 이 논문은 경제적인 솔루션으로써 마이크로 컨트롤러의 중요성이 대두되는 상황에서 마이크로 컨트롤러를 타겟으로한 어셈블러 포팅에 대한 지침을 제시함으로써 새로운 마이크로 컨트롤러의 진입장벽을 낮출 수 있을 것이다.

참고문헌

[1] 고상원, “주요 IT부품시장 동향분석 및 정책대안연구 : IT SoC를 중심으로,” 2006 ITFIND 동향자료, 2005년 6월

[2] 이희, “고성능 국산 임베디드 마이크로프로세서 (EISC),” 2006 한국통신학회, 한국통신학회지 제23권 제5호 2006년 5월

[3] ㈜에이디칩스, “4/8bit Microprocessor Instruction Set Manual,” 2011 ㈜에이디칩스, 2011

[4] ASxxxx Cross-Assemblers , http://shop-pdp.kent.edu/ashtml/asxxxx.htm