GCC RTL Representation
(1) 개요와 정의

정원교
weongyo@hotmail.com

2005년 9월 23일

목 차

제 1 절 16 주째 문서를 시작하며

이 문서는 GNU 컴파일러의 back end 내에서 사용되는 Register Transfer Language (rtl) 을 구성하는 Register Transfer Expressions (rtx's) 을 위한 정의와 문서들을 포함하고 있습니다.

제 2 절 RTX 정의및 설명

RTX 는 RTL 을 구성하는 구성요소로써 RTX 가 모여서 RTL 이 된다. 각 RTX 는 TREE 구조에서 변화한 모습으로 자리잡고 있으며, 포괄적으로 GCC 에서 지원하는 machine 들의 특성들을 통합적으로 포함할 수 있는 구조로 설계되어 있다. $prefix/gcc/rtl.def 파일에는 모든 target 들을 위한 expression 정의와 설명들이 있습니다. 몇몇은 어떤 target 에서는 사용되지 않을 수 있습니다.

2.1 전체적인 윤곽

아래의 표에서 "**형식**"에 관한 각 문자의 의미를 설명을 하면 다음과 같다. 각 문자는 rtx 내에서의 print format 과 각 rtx->fld[] (field) 의 type 을 표현을 하는데, 이 형식들은 전역변수 rtx_format[] 내에 저장된다.

* - 정의되어 있지 않음. 경고 메세지를 일으킬 수 있다.

0 - field 는 사용되지 않습니다. (혹은 phase-dependent manner 에서 사용될 수도 있습니다.) 아무것도 출력하지 않는다.

i - 정수. 정수를 출력한다.

n - "i" 와 비슷하지만, 'note_insn_name' 에서 entry 들을 출력한다.

w - HOST_BITS_PER_WIDE_INT 길이의 정수. 정수를 출력한다.

s - 문자열 포인터. 문자열을 출력한다.

S - "s" 와 비슷하지만, 선택적: Containing rtx 는 이 operand 전에 끝날 수 있다.

T - "s" 와 비슷하지만, RTL reader 에 의해 특별히 취급된다; 오직 machine description pattern 들에서만 찾을 수 있다.

e - rtl expression 포인터. expression 을 출력한다.

E - rtl 표현식들의 갯수를 가르키는 vector 로의 pointer rtl 표현식의 리스트를 출력한다.

V - "E" 와 비슷하지만, 선택적: Containing rtx 는 이 operand 전에 끝날 수 있다.

u - 다른 insn 로의 pointer. insn 의 uid 를 출력한다.

b - bitmap header 를 가르키는 pointer 이다.

t - tree 포인터이다.

아래의 리스트는 표의 rtx 의 "**분류**"를 나타낸다. 이것들은 rtx_class 내에 저장되고 GET_RTX_CLASS macro 를 통해서 접근된다. 그들은 다음과 같이 정의된다.

o 는 object (예를 들면, REG, MEM) 를 나타내는데 사용될 수 있는 rtx code

< 는 비교 (예를 들면, EQ, NE, LT) 를 나타내는데 사용될 수 있는 rtx code

1 는 unary 산술 표현식 (예를 들면, NEG, NOT) 를 나타내는데 사용될 수 있는 rtx code

c 는 commutative binary operation (예를 들면, PLUS, MULT) 를 나타내는데 사용될 수 있는 rtx code

3 는 non-bitfield three input operation (IF_THEN_ELSE) 를 나타내는데 사용될 수 있는 rtx code

2 는 non-commutative binary operation (예를 들면, MINUS, DIV) 를 나타내는데 사용될 수 있는 rtx code

b 는 mnbit-field operation (ZERO_EXTRACT, SIGN_EXTRACT) 를 나타내는데 사용될 수 있는 rtx code

i 는 machine insn (INSN, JUMP_INSN, CALL_INSN) 를 나타내는데 사용될 수 있는 rtx code

m 는 insn 들을 매칭하는 어떤 것을 (예를 들면, MATCH_DUP) 나타내는데 사용될 수 있는 rtx code

g 는 insn 들을 함께 그룹화할 때 (예를 들면, GROUP_PARALLEL) 사용될 수 있는 rtx code

a 는 autoincrement addressing mode 들 (예를 들면, POST_DEC) 를 나타내는데 사용될 수 있는 rtx code

x 는 다른 모든 것에 사용됨

다음 표는 각 RTX 에 대해 알파벳 순서로 정리한 표이다. 형식과 분류에 대해서는 위의 리스트를 참조하면 된다.

이름	형식	분류	이름	형식	분류
ABS	e	1	ADDR_DIFF_VEC	eEee0	x
ADDR_VEC	E	x	ADDRESS	e	m
ADDRESSOF	eit	o	AND	ee	c
ASHIFT	ee	2	ASHIFTRT	ee	2
ASM_INPUT	s	x	ASM_OPERANDS	ssiEEsi	x
ATTR_FLAG	s	x	ATTR	s	x
BARRIER	iuu	x	CALL_INSN	iuueieee	i
CALL_PLACEHOLDER	uuuu	x	CALL	ee	x
CC0		o	CLOBBER	e	x
CODE_LABEL	iuu00iss	x	COMPARE	ee	2
CONCAT	ee	o	COND_EXEC	ee	x
COND	Ee	x	CONST_DOUBLE	CDF[1]	o
CONST_INT	w	o	CONST_STRING	s	o
CONST_VECTOR	E	x	CONST	e	o
CONSTANT_P_RTX	e	x	DEFINE_ASM_ATTRIBUTES	V	x
DEFINE_ATTR	sse	x	DEFINE_COMBINE	Ess	x
DEFINE_COND_EXEC	Ess	x	DEFINE_DELAY	eE	x
DEFINE_EXPAND	sEss	x	DEFINE_FUNCTION_UNIT	siieiiV	x
DEFINE_INSN_AND_SPLIT	sEsTsESV	x	DEFINE_INSN	sEsTV	x
DEFINE_PEEPHOLE	EsTV	x	DEFINE_PEEPHOLE2	EsES	x
DEFINE_SPLIT	EsES	x	DIV	ee	2
EQ_ATTR	ss	x	EQ	ee	<
EXPR_LIST	ee	x	FFS	e	1
FIX	e	1	FLOAT_EXTEND	e	1
FLOAT_TRUNCATE	e	1	FLOAT	e	1
GE	ee	<	GEU	ee	<
GT	ee	<	GTU	ee	<
HIGH	e	o	IF_THEN_ELSE	eee	3
INCLUDE	s	x	INSN_LIST	ue	x
INSN	iuueiee	i	IOR	ee	c
JUMP_INSN	iuueiee0	i	LABEL_REF	u00	o
LE	ee	<	LEU	ee	<

[1] 여기서 CDF 는 CONST_DOUBLE_FORMAT 의 줄임말로 이 매크로는 각 target 에 특성에 따라 그 format 이 달라진다. 이에 대한 자세한 설명은 표의 끝 부분에서 하도록 하겠다.

LO_SUM	ee	o	LSHIFTRT	ee	2
LT	ee	<	LTGT	ee	<
LTU	ee	<	MATCH_DUP	i	m
MATCH_INSN	is	m	MATCH_OP_DUP	iE	m
MATCH_OPERAND	iss	m	MATCH_OPERATOR	isE	m
MATCH_PAR_DUP	iE	m	MATCH_PARALLEL	isE	m
MATCH_SCRATCH	is	m	MEM	e0	o
MINUS	ee	2	MOD	ee	2
MULT	ee	c	NE	ee	<
NEG	e	1	NIL	*	x
NOT	e	1	NOTE	iuu0ni	x
ORDERED	ee	<	PARALLEL	E	x
PC		o	PHI	E	x
PLUS	ee	c	POST_DEC	e	a
POST_INC	e	a	POST_MODIFY	ee	a
PRE_DEC	e	a	PRE_INC	e	a
PRE_MODIFY	ee	a	PREFETCH	eee	x
QUEUED	eeeee	x	RANGE_INFO	uuEiiiiiibbii	x
RANGE_LIVE	bi	x	RANGE_REG	iiiiiiiitt	x
RANGE_VAR	eti	x	REG	i0	o
RESX	i	x	RETURN		x
ROTATE	ee	2	ROTATERT	ee	2
SCRATCH	0	o	SEQUENCE	E	x
SET_ATTR_ALTERNATIVE	sE	x	SET_ATTR	ss	x
SET	ee	x	SIGN_EXTEND	e	1
SIGN_EXTRACT	eee	b	SMAX	ee	c
SMIN	ee	c	SQRT	e	1
SS_MINUS	ee	2	SS_PLUS	ee	c
SS_TRUNCATE	e	1	STRICT_LOW_PART	e	x
SUBREG	ei	x	SYMBOL_REF	s	o
TRAP_IF	ee	x	TRUNCATE	e	1
UDIV	ee	2	UMAX	ee	c
UMIN	ee	c	UMOD	ee	2
UNEQ	ee	<	UNGE	ee	<
UNGT	ee	<	UNKNOWN	*	x
UNLE	ee	<	UNLT	ee	<
UNORDERED	ee	<	UNSIGNED_FIX	e	1
UNSIGNED_FLOAT	e	1	UNSPEC_VOLATILE	Ei	x
UNSPEC	Ei	x	US_MINUS	ee	2
US_PLUS	ee	c	US_TRUNCATE	e	1
USE	e	x	VALUE	0	o
VEC_CONCAT	ee	x	VEC_DUPLICATE	e	x
VEC_MERGE	eee	x	VEC_SELECT	ee	x
XOR	ee	c	ZERO_EXTEND	e	1
ZERO_EXTRACT	eee	b			

위의 각주로 있는 CDF (CONST_DOUBLE_FORMAT 의 줄임말) 는 $prefix/gcc/rtl.c 파일의 앞부분에 선언되어 있는 매크로에 의해 각 시스템의 특성에 따라 상대적으로 형식이 변화하게 되어 있다. 이 값은 실질적으로 매크로 REAL_WIDTH 의 값에 따라 변화하는데, 이 매크로에 영향을 주는 다른 매크로로는 REAL_ARITHMETIC, MAX_LONG_DOUBLE_TYPE_SIZE, HOST_BITS_PER_WIDE_INT 등등이 존재한다. 자세한 작동에 대해서는 이 rtl.c 파일을 참조하면 쉽게 알 수 있다.

2.2 각 rtx 에 대해

아래의 설명의 각 주제들은 공백문자로 나눌 경우, 첫번째 인자는 rtx 의 이름, 두번째는 형식, 세번째는 분류이다. 그리고 각 성격에 따라 페이지를 나누었다.

프로그램의 rtl 표현식을 구조화하는데 사용되는 표현식들 (그리고 “meta” 표현식들).

UNKNOWN * x

expression code 이름이 reader 에서 알 수 없는 것임을 나타냄.

NIL * x

(NIL) 는 null 포인터를 나타내는데 rtl reader 와 printer 에 의해 사용됩니다.

INCLUDE s x

파일 include

list 들을 구성하는데 사용되는 expression 들.

EXPR_LIST ee x

expression 들의 연결 리스트.

INSN_LIST ue x

instruction 들의 연결 리스트.
명령어들은 그들의 uid 들을 출력함으로써 표현되어 진다.

machine description 들을 위한 표현식 type 들. 이것들은 컴파일러내의 실제 rtl code 에서는 나타나지 않는다.

MATCH_OPERAND iss m

오직 machine description 들에서만 나타남.
매개체들 (means) 은 두번째 arg (문자열) 로 명명된 함수를 술어로써 사용한다; 만약 매치된 었다면, 구조체를 저장한다. 이 구조체는 첫번째 arg (정수) 에 지정된 index 에 있는 operand table 내에 매치된 것이다. 만약 두번째 arg 가 null 문자열이라면, 구조체는 단순히 저장된다.

세번째 문자열 인자는 operand 가 할당될 수 있는 곳에 대한 register allocator 제한을 나타낸다.

만약 target 이 어떠한 명령어에 대한 제한이 필요하지 않으면 이 field 는 반드시 null 문자열이여야 한다.

문자열은 다음과 같이 준비하였다:
'=' 는 operand 가 오직 쓰기만 가능함을 나타냄.
'+' 는 operand 가 읽기, 쓰기 양쪽다 가능함을 나타냄.

문자열의 각 문자는 operand 를 위한 할당가능한 class 를 나타낸다.
'g' 는 operand 가 어떠한 값의 class 이어도 됨을 나타낸다.
'i' 는 operand 가 (명령어에서) immediate data 가 될수 있음을 나타낸다.
'r' 는 operand 가 register 내에 있을 수 있음을 나타낸다.
'm' 는 operand 가 메모리 내에 있을 수 있음을 나타낸다.
'm' class 의 'o' 부분집합. 그 memory addressing mode 들은 컴파일시 offset 일수 있다. (그들에게 더해진 상수를 가지고 있음)

다른 문자들은 target dependent operand class 들을 나타내고 각 target 의 machine description 내에 표현되어 있다.

하나의 operand 이상을 가지는 명령어들에 대해, class 들의 집합들은 적당한 multi-operand constraint 들을 가르키는 comma (콤마 ,) 로 구분될 수 있다. 명령어에 대한 모든 operand 내의 이 class 들의 집합들 사이에서는 반드시 1 대 1 대응해야 한다.

MATCH_SCRATCH is m

오직 machine description 들에서만 나타남.
매개체들은 SCRATCH 혹은 register 에 매치한다. rtl 를 생성하기 위해 사용될 때, SCRATCH 가 생성된다. MATCH_OPERAND 로써는 mode 가 요구된 mode 를 지정하고 첫번째 인자는 operand 번호이다. 두번째 인자는 constraint 이다.

MATCH_DUP i m

오직 machine description 들에서만 나타남.
매개체들은 오직 인자로 지정된 index 의 operand table 내에 저장되어 있는 어떤 것과 같은 것을 매치한다.

MATCH_OPERATOR isE m

오직 machine description 들에서만 나타남.
매개체들은 술어를 적용하며, AND 는 rtx 의 operand 들을 재귀적으로 매치한다.
Operand 0 은 match_operand 내와 같은 operand-번호 이다.
Operand 1 은 (문자열, 함수 이름으로써) 적용할 술어이다.
Operand 2 는 표현식들의 vector, 각각의 것들은 반드시 이 construct 가 매칭 하는 rtx 의 한 하위표현식과 매치해야 한다.

MATCH_PARALLEL isE m

오직 machine description 들에서만 나타남.
임의의 길이의 PARALLEL 에 매치되는 매개체들. 술어는 PARALLEL 와 PARALLEL 가 매치된 초기의 표현식들에 적용된다.
Operand 0 는 match_operand 내와 같은 operand-번호.
Operand 1 는 PARALLEL 에 적용할 술어이다.
Operand 2 는 표현식들의 vector, 각각의 것들은 반드시 PARALLEL 내에 대응하는 요소에 매치되어야 한다.

MATCH_OP_DUP iE m

오직 machine description 들에서만 나타남.
매개체들은 인자로 지정된 index 인 operand table 내에 저장되어 있는 어떤 것과 같은 것에만 매치한다. MATCH_OPERATOR 용.

MATCH_PAR_DUP iE m

오직 machine description 들에서만 나타남.
매개체들은 인자로 지정된 index 인 operand table 내에 저장되어 있는 어떤 것과 같은 것에만 매치한다. MATCH_PARALLEL 용.

MATCH_INSN is m

오직 machine description 들에서만 나타남.
Operand 0 는 match_operand 내와 같은 operand-번호.
Operand 1 는 insn 에 적용할 술어이다.

DEFINE_INSN sEsTV x

오직 machine description 들에서만 나타남.
명령어의 한 종류를 위한 패턴을 정의한다.

```
Operand:
0: 이 명령어를 명명한다.
   만약 이름이 null 문자열이면, 명령어는 단지 machine description 에서
   인식만 될 뿐이지, tree 에서 rtl 로의 확장자에게 넘겨지지 않는다.
1: 는 패턴이다.
2: 는 문자열인데, 이 문자열은 이 패턴을 인식하기 위한 부과적인 조건을 주는
   C 표현식이다.
   NULL 문자열은 여분의 조건이 없음을 의미한다.
3: 는 만약 패턴이 매치된다면 실행할 행동이다.
   만약 이 어셈블러 code template 이 * 로 시작한다면 사용할 template 을 결정하기
   위해 실행할 C code 의 한 부분이다.  그렇지 않다면 그것은 사용할 template 이다.
4: 선택적으로, 이 명령어를 위한 attribute 들의 vector.
```

DEFINE_PEEPHOLE EsTV x

Peephole 최적화의 정의.
첫번째 operand : 매치를 위한 insn 패턴들의 vector
두번째 operand : 반드시 true 여야 하는 C 표현식
세번째 operand : 어셈블러 output 를 생성하기 위한 template 혹은 C code
네번째 operand : 선택적인, 이 insn 을 위한 attribute 들의 vector

DEFINE_SPLIT EsES x

Split operation 의 정의.

```
1st operand: 매치를 위한 insn 패턴
2nd operand: 반드시 true 여야 하는 C 표현식
3rd operand: SEQUENCE 내에 자리잡을 insn 패턴들의 vector
4th operand: 선택적인, insn 들을 생성하기 전에 실행할 몇몇 C code.  예를 들면,
    이것은 몇몇 RTX 들을 생성하고 insn-패턴들의 vector 에 의해 사용되기 위해
    'recog_data.operand' 의 element 들 속에 그들을 저장할 것이다.
    (여기서 'operands' 는 'recog_data.operand' 토의 alias 이다.).
```

DEFINE_INSN_AND_SPLIT sEsTsESV x

Insn 와 associated split 의 정의
이것은 몇몇 수정부분들을 가지는 define_insn 와 define_split 의 연속이며 이것들은 같은 패턴을 공유한다.

```
Operand:
0: 이 명령어를 명명한다.
   만약 이름이 null 문자열이면, 명령어는 단지 machine description 에서
   인식만 될 뿐이지, tree 에서 rtl 토의 확장자에게 넘겨지지 않는다.
1: 는 패턴이다.
2: 는 문자열인데, 이 문자열은 이 패턴을 인식하기 위한 부과적인 조건을 주는
   C 표현식이다.
   NULL 문자열은 여분의 조건이 없음을 의미한다.
3: 는 만약 패턴이 매치된다면 실행할 행동이다.
   만약 이 어셈블러 code template 이 * 토 시작한다면 사용할 template 을 결정하기
   위해 실행할 C code 의 한 부분이다.  그렇지 않다면 그것은 사용할 template 이다.
4: Split 에 대해 반드시 true 여야 하는 C 표현식.  이것은 split condition 이
   어떤 경우에 insn condition 의 logical and 인 경우라던가, 이 operand 의
   "&&" 에 무엇이 뒤에 따르냐에 따라 "&&" 으로 시작될 수 있다.
5: SEQUENCE 내에 자리잡을 insn 패턴들의 vector
6: 선택적으로, insn 들을 생성하기 전에 실행할 몇몇 C code.  예를 들면,
   이것은 몇몇 RTX 들을 생성하고 insn-패턴들의 vector 에 의해 사용되기 위해
   'recog_data.operand' 의 element 들 속에 그들을 저장할 것이다.
   (여기서 'operands' 는 'recog_data.operand' 토의 alias 이다.).
7: 선택적으로,  이 insn 을 위한 attribute 들의 vector.
```

DEFINE_PEEPHOLE2 EsES x

RTL peephole operation 의 정의.
define_split 와 같은 인자들이 따른다.

DEFINE_COMBINE Ess x

Combiner pattern 의 정의.
Operand 들이 아직 정의되지 않았다.

DEFINE_EXPAND sEss x

표준 insn 이름을 위한 다수의 insn 들을 어떻게 생성할지를 정의한다.

```
1st operand: insn 이름.
2nd operand: insn-pattern 들의 vector.
```

‘recog_data.operand’ 의 element 를 대체하기 위해 match_operand 를 사용한다.
3rd operand: 이용가능하기 위해 이것을 위해 반드시 true 여야하는 C 표현식.
이것은 어떠한 operand 들도 검사하지 않을 것이다.
4th operand: insn 들을 생성하기 전에 실행할 여분의 C code. 예를 들면,
이것은 몇몇 RTX 들을 생성하고 insn-패턴들의 vector 에 의해 사용되기 위해
‘recog_data.operand’ 의 element 들 속에 그들을 저장할 것이다.
(여기서 ‘operands’ 는 ‘recog_data.operand’ 토의 alias 이다.).

DEFINE_DELAY eE x

Delay slot 들을 위한 requirement 를 정의한다.

1st operand: insn attribute 들을 포함하는 조건, 만약 true 일 경우, 이것은
insn 가 나타난 delay slot 들의 갯수를 요구한다는 것을 가르킨다.
2nd operand: Vector 를 나타내는데, 이것의 길이는 요청된 delay slot 들의
갯수보다 3 배이다.
각 entry 는 세가지 조건들을 있으며, 각각은 attribute 들을 포함한다.
첫번째는 delay slot location 을 차지하기 위한 insn 에 대해 반드시
참이여야 한다. 두번째는 모든 명령어들에 대해 true 인데, 만약 branch
가 true 일경우 무효토 될수 있고, 세번째도 모든 insn 들에 대해 true
이지만 bracnch 가 false 일 경우 무효토 될 수 있다.

다수의 DEFINE_DELAY 들이 존재할 수 있다. 그들은 delay slot 들에 대한 각기 다른 요구 사항들을 가르킨다.

DEFINE_FUNCTION_UNIT siieiiV x

함수 unit 에 필요한 insn 들의 집합을 정의한다. 이것은 이 insn 들이 delay 후에 그들의 결과를 생성함을 의미하고 동시적으로 스케줄될 수 있는 이 type 의 insn 들의 갯수상에 제한이 있을 수 있음을 의미한다.

한 DEFINE_FUNCTION_UNIT 보다 많이 함수 unit 을 위해 지정될 수 있다. 각각은 operation 들의 집합과 associated delay 들을 가지고 있다. 처음 3 개의 operand 들은 같은 함수 unit 을 위해서 각 operation 에 대해 반드시 같아야만 한다.

모든 delay 들은 cycle 들내에 지정된다.

1st operand: 함수 unit (대부분 문서화를 위해) 의 이름
2nd operand: CPU 내 동일한 함수 unit 들의 갯수
3rd operand: 이 함수 unit 상에서 실행할 수 있는 동시적인 insn 들의 최
갯수; 만약 제한이 없다면 0.
4th operand: insn attribute 를 포함하는 조건, 만약 true 라면, 이 attribute
는 이 표현식을 적용할 저 insn 들을 지정한다.
5th operand: 어떤 insn 결과가 이용가능한지 따진 후 constant delay
6th operand: 다음 insn 까지의 delay 는 이 operation 을 실행할 함수 unit 상에
스케줄되어질 수 있다. 의미는 다음 operand 가 제공되어 지는냐
아니냐에 달려있다.
7th operand: 만약 이 operand 가 지정되어 있지 않는다면 4th operand 와 매치
하는 명령어가 하위의 다음 insn 를 시작할 수 있을 때까지 함수
unit 을 사용하기 시작한 후, 6th operand 가 cycle 들의 수를
준다. 값 0 는 issue constraint 들을 가지고 있지 않은
unit 에 대해 사용되어져야 한다. 만약 오직 한 operation 만이
한번에 실행될 수 있고 unit 이 전체 시간동안 바쁠 경우,
3rd operand 가 1 토써 지정되어야 하고, 6th operand 는 0 으토

지정되어야 한다. 그리고 7th operand 는 지정되어서는 안된다.

만약 이 operand 가 지정되었다면, 그것은 attribute 표현식들의 리스트이다. 만약 그러한 어떤 표현식들이 true 인 insn 가 현재 함수 unit 상에서 실행중이라면, issue delay 는 6th operand 에 의해 주어질 것이다. 그렇지 않다면, insn 는 즉시 스케줄 되어질 수 있다. (Unit 상에서 실행되어질 수 있는 동시적인 operation 들의 갯수상의 제한에 대한 속성).

DEFINE_ASM_ATTRIBUTES V x

'asm' 명령어들을 위한 attribute computation 를 정의한다.

DEFINE_COND_EXEC Ess x

conditional execution meta operation 의 정의. 자동적으로 DEFINE_INSN 의 새로운 instance 들을 생성하는데, 이 DEFINE_INSN 는 attribute "predicable" true 를 가진 것으로 선택되었다. 새로운 패턴은 COND_EXEC 와 top-level 에서의 속성을 포함할 것이다.

Operand:
0: 속성(술어) 패턴. top-level form 은 relational operator 와 매치되어야 한다. Operand 들은 오직 하나의 alternative 를 가져야 한다.
1: 생성된 패턴을 인식하기 위한 부과적인 조건을 담고 있는 C 표현식
2: 어셈블러 output 을 생성하기 위한 template 혹은 C code.

SEQUENCE E x

SEQUENCE 는 여러 insn 들을 만들기 원하는 DEFINE_EXPAND 에 대한 'gen_...' 함수의 결과에서 나타난다. 그것의 element 들은 만들어져야 하는 insn 들의 body 들이다. 'emit_insn' 는 SEQUENCE 의 부분을 가지고 separate insn 들을 만든다.

ADDRESS e m

그것의 인자의 주소를 참조한다. 이것은 오직 alias.c 에서만 사용된다.

Insn attribute 들에 사용되는 표현식들. 이것들 또한 컴파일러내 실제 rtl code 에서 나타나지 않는다.

DEFINE_ATTR sse x

Insn attribute 의 정의.
1st operand: attribute 의 이름
2nd operand: 가능한 attribute 값들의 콤마로-분리된 목록
3rd operand: attribute 의 기본값을 위한 표현식

ATTR s x

Attribute 의 이름을 위한 marker.

SET_ATTR ss x

패턴이 매칭하는 insn 들을 할당하기 위한 attribute 를 지정하는 DEFINE_ASM_INSN 내에서의 사용을 위해서라던가, DEFINE_INSN 혹은 DEFINE_PEEPHOLE 의 마지막 (선택적인) operand 내의 사용에 대해.

```
(set_attr "name" "value") 는
  (set (attr "name") (const_string "value")) 와 같다.
```

SET_ATTR_ALTERNATIVE sE x

DEFINE_INSN 와 DEFINE_PEEPHOLE 의 마지막 operand 에서, 이것은 attribute 값들이 alternative match 되는 것에 따라 할당되는 것이라고 지정하는데 사용될 수 있다.

다음 세가지 표현식은 같다:

```
(set (attr "att") (cond [(eq_attrq "alternative" "1") (const_string "a1")
                         (eq_attrq "alternative" "2") (const_string "a2")]
                        (const_string "a3")))
(set_attr_alternative "att" [(const_string "a1") (const_string "a2")
                             (const_string "a3")])
(set_attr "att" "a1,a2,a3")
```

EQ_ATTR ss x

만약 현재 insn 의 지정된 attribute 의 값이 지정된 값과 같으면 conditional 표현식이 true 이다. 첫번째 operand 는 attribute 이름이고 두번째는 comparison 값이다.

ATTR_FLAG s x

만약 지정된 flag 가 reorg 내에 스케줄되어진 insn 에 대해 true 일 경우 true 인 conditional 표현식.

genattr.c 는 다음의 flag 들을 정의하는데, 이 flag 들은 eligible_for_delay 내 (attr_flag “foo”) 표현식들에 의해 테스트될 수 있다.

forward 와 backward, very_likely, likely, very_unlikely, unlikely

명령어 chain 내의 것들에서 사용되어지는 표현식 type 들.
chain 를 다루기 위해 모든 format 들은 “iuu” 로 반드시 시작되어야 한다. 각 insn 표현식은 back-end 처리 기간 동안 rtl 명령어와 그것의 semantic 들을 가지고 있다. 각 rtx->fld[] 의 의미를 이해하기 위해서는 “rtl.h” 내의 매크로들을 보라.

INSN iuueiee i

Jump 할 수 없는 명령어.

JUMP_INSN iuueiee0 i

Jump 가 가능할 수 있는 명령어 Field 들 (rtx->fld[]) 은 INSN 들과 똑같은 의미를 가지고 있다.

CALL_INSN iuueieee i

Subroutine 를 호출할 가능성이 있는 명령어, 하지만 명령어가 현재 함수 내에 다음에 올 수 있음이 변하지 않을 것으로. Field (rtx->fld[7]) 는 CALL_INSN_FUNCTION_USAGE 이다. 모든 다른 field 들 (rtx->fld[]) 는 INSN 들과 완전히 같은 의미를 가진다.

BARRIER iuu x

Control 이 어떤 것을 통해 흐르지 않을 것임을 나타내는 marker.

CODE_LABEL iuu00iss x

명령어에 따른 label 를 가지고 있다.

```
Operand:
3: 는 label 의 use-count 를 위해 jump.c 에서 사용된다.
4: 는 이 label 로의 label_ref 들의 chain 을 가르키기 위해 flow.c 에서 사용된다.
5: 는 전체 컴파일 과정에서 유일한 번호이다.
6: 는 어찌하였든, label 의 user-given name 이다.
7: 는 alternate label name 이다.
```

NOTE iuu0ni x

symbol table 을 위해 code 내에서 source line 이 어디서 시작하는지 말해준다.

```
Operand:
3: 만약 줄 번호 $>$ 0 이면, 파일이름, 그렇지 않으면 note-specific data.
4: 만약 0 보다 크면 줄번호, 그렇지 않으면 enum note_insn.
5: 만약 줄번호 == note_insn_deleted_label 이면 unique 번호.
```

INSN 와 JUMP_INSN, CALL_INSN 의 top level 구성 요소.

COND_EXEC ee x

조건적인 execute code.
Operand 0 는 condition 이며, true 일 경우, code 는 실행된다.
Operand 1 는 실행될 code 이다. (전형적인 SET).

Semantic 들은 만약 condition 이 false 일 경우 부과적인 영향을 가지고 있지 않다. 이 패턴은 reload 실행 후 if_convert 단계에 의해서나 target-specific splitter 들에 의해 자동으로 생성된다.

PARALLEL E x

병렬로 처리되는 여러 operation 들 (아마도 COND_EXEC 내에서).

ASM_INPUT s x

Input 으로 어셈블러에게 건네진 문자열. 어셈블러 comment 문법을 사용함으로써 comment 들을 분명하게 전달 할 수 있다. 이것들은 insn 내에서 발생할 수 있는데, PATTERN 으로써 그들 자신에게도 해당된다. 그들은 또한 문자열을 잡기 위한 편리한 방식으로써 ASM_OPERANDS 내에 나타날 수 있다.

ASM_OPERANDS ssiEEsi x

Operand 들을 가진 어셈블러 명령어.

```
1st operand 는 명령어 template 이다.
2nd operand 는 output 을 위한 제약조건이다.
3rd operand 는 이 표현식이 참고하는 output 의 갯수이다.
    Insn 가 하나의 값 이상을 저장할 때, separate ASM_OPERANDS 는
    각 output 에 대해 만들어진다; 이 정수는 그들과 구별된다.
4th 는 input operand 값들의 vector 이다.
5th 는 input operand 들을 위한 mode 들과 제약조건들의 vector 이다.
    각 element 는 제약조건 문자열을 포함하는 ASM_INPUT 이고 그러한
    것의 mode 는 input operand 의 mode 를 가르킨다.
6th 는 포함하고 있는 source file 의 이름이다.
7th 는 source line 번호이다.
```

UNSPEC Ei x

Machine-specific operation 1st operand 는 operation 에 의해 사용되는 operand 들의 vector 이다. 그래서 reload 들에 필요한 것들이 수행될 수 있다.
2nd operand 는 machine-specific operation 의 어떤 번호가 실행되어 진것인지 를 말해주는 unique 값이다. (genrecog.c record 는 insn 내에 자리잡고 있는 방식으로 인해서 vector 는 반드시 첫번째 operand 여야만 함을 알기 바란다.) 이것은 PATTERN 내에서 그 자신을 포함하여 PARALLEL 의 component 로써 혹은 표현식 내부에서 모두 발생할 수 있다.

UNSPEC_VOLATILE Ei x

비슷하지만, volatile operation 이고 trap 일 수 있다.

ADDR_VEC E x

주소들의 vector, 전체 단어로써 저장된다.

각 element 는 우리가 원하는 주소인 CODE_LABEL 로의 LABEL_REF 이다.

ADDR_DIFF_VEC eEee0 x

주소 부분의 다른점에 관한 vecotr X0 - BASE, X1 - BASE, ...
첫번째 operand 는 BASE 이다; Vector 는 X 들을 포함하고 있다. 이 rtx 의 machine mode 는 각 다른점에 대해 얼마나 많은 공간을 남겨두고 있는지 말해주고 만약
CASE_VECTOR_SHORTEN_MODE 가 정의되어 있을 경우 brach shortening 에 의해 적용된다.
세번째 와 네번째 operand 들은 저마다의 최소와 최대 주소들을 가진 target label 들을 저장한다.
다섯번째 operand 는 branch shortening 에서 사용될 flag 들을 저장한다.

shorten_branches 의 시작 부분에 설정:
min_align: 어떤 target label 들의 최소 alignment.
base_after_vec: 만약 BASE 가 ADDR_DIFF_VEC 뒤에 있다면 true.
min_after_vec: 만약 최소 addr target label 이 ADDR_DIFF_VEC 뒤에 있다면 true.
max_after_vec: 만약 최대 addr target label 이 ADDR_DIFF_VEC 뒤에 있다면 true.
min_after_base: 만약 최소 address target label 가 BASE 뒤에 있다면 true.
max_after_base: 만약 최대 address target label 가 BASE 뒤에 있다면 true.
실제 branch shortening 과정에서 설정:
offset_unsigned: 만약 offset 들이 unsigned 으로써 취급되어야 한다면 true.
scale: Mode 에 맞게 offset 들을 만드는데 필요한 scaling.

세번째와 네번째, 다섯번째 operand 들은 CASE_VECTOR_SHORTEN_MODE 가 정의되었을 때와 optimizing compilation 시에만 유효하다.

PREFETCH eee x

몇몇 target 들상에서 지원되는 attribute 들을 가진 메모리 prefetch. Operand 1 는 fetch 할 메모리의 주소이다.
Operand 2 는 write access 시 1, 그렇지 않을 경우 0 이다.
Operand 3 는 temporal locality 의 level 이다; 0 은 temporal locality 가 없음을 의미하고, 1 과 2, 3 은 temporal locality 의 level 이 증가했음을 의미한다.

Operand 2 와 3 에 의해 지정된 attribute 들은 target 의 prefetch 명령어들이 그들을 지원하지 않는 것에 대해서는 무시된다.

명령어의 top level 에서. (아마도 PARALLEL 하위에서).

SET ee x

> Assignment.
> Operand 1 은 지정된 위치 (REG, MEM, PC, CC0 혹은 어딘가).
> Operand 2 는 거기 저장된 값.
> 모든 assignment 는 반드시 SET 을 사용해야 한다. 다수의 assignment 들을 수행하는 명령어들은 PARALLEL 하에서는 반드시 다수의 SET 을 사용해야 한다.

USE e x

> 우리가 설명하기를 원하지 않는 방식으로 어떤 것이 use 됨을 나타낸다. 예를 들면, subroutine call 들은 static chiain 이 건내졌었던 register 를 use 할 것이다.

CLOBBER e x

> 우리가 설명하기를 원하지 않는 방식으로 어떤 것이 clobber 됨을 나타낸다. 예를 들면, subroutine call 들은 몇몇 물리적 register 들을 clobber 할 것이다. (편리함에 의해 저장되지 않는 어떤 것들)

CALL ee x

> subroutine 을 호출(call)합니다.
> Operand 1 은 호출할 주소.
> Operand 2 는 argument 들의 수.

RETURN x

> subroutine 으로부터 return.

TRAP_IF ee x

> Conditional trap.
> Operand 1 는 condition 이다.
> Operand 2 는 trap code 이다.
> unconditional trap 를 위해, condition (const_int 1) 를 만든다.

RESX i x

> 함수 call 혹은 branch 가 필요한지 아닌지를 우리가 알기 전에 _Unwind_Resume 를 위한 공간 (장소). Operand 1 는 어디에서 부터 control 이 흐르고 있는지에 대한 exception region 이다.

Expression 들에서 사용되는 원시값 (primitive value).

CONST_INT w o

숫자인 정수 상수

CONST_DOUBLE CONST_DOUBLE_FORMAT o

숫자인 부동소수점 (실수) 상수.
Operand 0 ('0') 은 현재 함수내에서 사용하는 모든 CONST_DOUBLE 들의 chain 입니다.
나머지 operand 들은 실제값을 가지고 있습니다. 그들은 모두 'w' 이며 아마 1 부터 4 까지 있을 것입니다; rtl.c 파일을 참조하십시오.

CONST_VECTOR E x

Vector 상수를 표현합니다.

CONST_STRING s o

문자열 상수. Attribute 들을 위해서만 사용된다.

CONST e o

이것은 표현식을 캡슐화하는데 사용되는데, 이 표현식의 값은 상수 (SYMBOL_REF 와 CONST_INT 의 합계와 같은) 이여서 산술 명령어들로 의해서 보다는 상수 operand 로써 인식될 것이다.

PC o

Program counter, 정상적인 jump 들은 처음 인자를 (PC) 로 가지는 SET 에 의해 표현됩니다.

VALUE 0 o

값을 표현하기 위해 cselib routine 들 내에서 사용된다.

REG i0 o

레지스터. "operand" 는 REGNO 매크로로 접근될 수 있는 레지스터 번호 (register number) 입니다. 만약 이 number 가 FIRST_PSEUDO_REGISTER 보다 작다면 hardware register 가 참고됩니다. 두번째 operand 는 원래 register number 를 잡고 있습니다 - 이것은 hard register 로 변한 것을 가지고 있는 pseudo register 와는 다를 것이다. 이 rtx 는 MEM 과 같은 가능한 많은 (더 많은) field 를 가지는 것을 필요로 합니다. 이것은 reload 동안 REG rtx 들을 MEM 들로 변경할 수 있기 때문입니다.

SCRATCH 0 o

Scratch register. 이것은 단일 insn 내에서만 사용되는 register 를 표현한다. 이것은 만약 제약 조건이 레지스터를 필요로 하거나, SCRATCH 를 남겨둘 수 있는 어떤 경우가 아니라면 register allocation 혹은 reload 동안 REG 로 변할 것이다. 이 code 는 하나의 operand 를 가진 것으로 mark 될 것이고 그래서 이것은 REG 로 변환될 수 있다.

SUBREG ei x

Multi-word 값의 한 word. 첫번째 operand 는 complete 값이다; 두번째가 어떤 word 인지 말한다. WORDS_BIG_ENDIAN flag 는 word number 0 (SUBREG 내에서 세어진 것으로써) 이 가장 최상의 혹은 만족하는 충분한 word 인지 아닌지를 제어한다.

이것은 또한 다른 machine mode 내에서 값을 참조하는데 사용된다. 예를 들면, SImode 값을 마치 그것이 Qimode 인것 처럼 혹은 그 반대로 참조하는데 사용될 수 있다. 그럴땐 word number 가 항상 0 이다.

STRICT_LOW_PART e x

이 하나의-인자인 rtx 는 move 명령어들을 위해 사용되는데, 그것은 목적지의 low part 만 변화하는 것을 보증받는 것만 해당된다. 그래서, (SET (SUBREG:HI (REG...)) (MEM:HI ...)) 는 REG 의 high part 상에 지정되지 않는 영향을 가지고 있지만, (SET (STRICT_LOW_PART (SUBREG:HI (REG...))) (MEM:HI ...)) 는 HImode 인 REG 의 오직 bit 들만 변화시킨다는 보증이 된다.

사용되는 실제 명령어는 아마도 두 경우에서 같을 것이지만, register 제약조건들은 STRICT_LOW_PART 가 사용중일 때는 좀 더 빡빡할 것이다.

CONCAT ee o

(CONCAT a b) 는 a 와 b 를 함께 놓을 수 있는 값을 만들기 위한 a 와 b 의 가상 연결 (concatenation) 을 나타낸다. 이것은 복소수값들을 위해 사용된다. 보통 DECL_RTL 들과 RTL 생성동안에만 나타나지, insn chain 에서는 나타나지 않는다.

MEM e0 o

메모리 위치; operand 는 주소이다. 두번째 operand 는 이 MEM 에 연결되어 있는 alias set 이다. 우리는 이 field 를 위해 'w' 대신 '0' 을 사용하였는데 이유는 이 field 는 machine description 에 지정될 필요가 없기 때문이다.

LABEL_REF u00 o

이 함수를 위한 code 내 assembler label 로의 참조. Operand 는 insn chain 에서 발견된 CODE_LABEL 이다. 프린트되지 않은 field 들 1 과 2 는 LABEL_NEXTREF 와 CONTAINING_INSN 를 위해 flow.c 에서 사용된다.

SYMBOL_REF s o

명명된(named) label 로의 참조: 처음 operand 는 문자열인데 묵시적으로 앞에 '_' 가 더한 형태입니다. 예외사항: 만약 처음 문자가 명시적으로 '*' 가 주어졌다면 그것을 어셈블러에게 줄 때 '*' 를 제거하지만 '_' 를 더하지는 않습니다.

CC0 o

Condition code register 는 우리들의 상상속에 0 과 비교될 수 있는 값을 가지고 있는 register 로써 표현됩니다. 사실 machine 는 이미 그것을 비교했고 결과를 기록했습니다; 하지만 condition code 를 살피는 명령어들은 전체적인 값을 살피고 그것을 비교하는데 사용되는 경향이 있습니다.

ADDRESSOF eit o

Register 의 address 를 참조 (reference). CSE 가 가능한 많이 생략한 후에 purge_addressof 에 의해 제거된다.
첫번째 operand: address of 에 필요할 수 있는 register.
두번째 operand: 목적을 위해 생성되었던 원래의 pseudo regno.
세번째 operand: put_reg_in_stack 용, register 내 object 를 위한 decl.

QUEUED eeeee x

QUEUED 표현식은 실제적으로 postincrement/postdecrement 에 대해 나중에 output 하기 위해서 명령어들의 queue 의 member 를 가르키고 있다. QUEUED 표현식들은 명령어들의 부분이 절대 되지 않는다. QUEUED vygustr이 명령어내에 놓여졌을 경우, 대신 증가된 변수 혹은 그것의 이전 값이 사용된다.

Operand 들은 다음과 같다:
0. 증가되어질 (REG rtx) 변수.
1. 증가 명령어 혹은 만약 그것이 아직 output 되지 않았을 경우 0.
2. 변수의 이전 값의 복사본인 REG rtx 혹은 아직 없다면 0.
3. 증가 명령어를 사용할 body.
4. Queue 내의 다음 QUEUED 표현식.

Rtl pattern 내의 operator 들을 위한 표현식들.

IF_THEN_ELSE eee 3

if_then_else. 이것은 보통의 conditional jump 명령어들에서 사용된다.

```
Operand:
0:  condition
1:  then expr
2:  else expr
```

COND Ee x

일반적인 conditional. 첫번째 operand 는 표현식의 여러 쌍으로 구성된 vector 이다. 각 쌍의 첫번째 element 는 차례로 감정되었다. conditional 의 값은 첫번째 쌍의 두번째 표현식이며 이 쌍의 두번째 표현식의 첫번째 표현식은 0 이 아님 감정한다. 만약 표현식들의 어느 것도 true 가 아니라면, 두번째 operand 는 conditional 의 값으로써 사용될 것이다.

이것은 IF_THEN_ELSE 의 사용에 대체되어야 한다.

COMPARE ee 2

비교, condition code 결과를 생성한다.

PLUS ee c

더하기

MINUS ee 2

Operand 0 에서 operand 1 을 빼기.

NEG e 1

Minus operand 0.

MULT ee c

곱하기

DIV ee 2

Operand 0 를 operand 1 로 나누기.

MOD ee 2

Operand 0 을 operand 1 로 나눈 후의 나머지.

UDIV ee 2
UMOD ee 2

Unsigned divide 와 remainder.

AND ee c
IOR ee c
XOR ee c
NOT e 1

Bitwise operation 들.

ASHIFT ee 2
ROTATE ee 2
ASHIFTRT ee 2
LSHIFTRT ee 2
ROTATERT ee 2

SHIFT 들.

```
Operand:
0:  shift 되는 값.
1:  bit 들의 갯수.
```

SMIN ee c
SMAX ee c
UMIN ee c
UMAX ee c

두 operand 들의 최소와 최대값들. 우리는 signed 와 unsigned form 두가지 가 필요하다. (우리는 SMIN 를 위해 MIN 을 사용할 수 없는데, 그것은 같은 이름의 매크로와 충돌하기 때문이다.

PRE_DEC e a
PRE_INC e a
POST_DEC e a
POST_INC e a

이 unary operation 들은 그들이 메모리 주소들에서 나타나는 것과 같이 증가와 감소를 나타내는데 사용된다. 증가와 감수의 양은 나타나지 않는데, 그들이 containing MEM 의 machine-mode 로 부터 협정된 것일 수 있기 때문이다. 이 operation 들은 오직 두 경우에만 존재한다:
1. stack 상에 push 할 경우.
2. flow.c 내 life_analysis 단계에서 자동으로 생성될 경우.

PRE_MODIFY ee a POST_MODIFY ee a

이 binary operation 들은 위의 operation 들을 사용하는 간단한 증가 혹은 감소를 제외하고 메모리 주소들내의 일반적인 address side-effect 들을 나타내는데 사용된다. 이것들은 flow.c 내 life_analysis 단계에 의해 자동으로 생성된다.
첫번째 operand 는 address 로 사용되는 REG 이다.
두번째 operand 는 address 를 평가하는 (PRE_MODIFY) 앞 혹은 (POST_MODIFY) 뒤 register 에 할당되어진 표현식이다.
현재, 컴파일러는 PLUS 의 첫번째 operand 가 *_MODIFY 의 첫번째 operand 와 같은 register 를 가져야 하는 장소인 (plus (reg) (reg)) 와 (plus (reg) (const_int)) 형식의 두번째 operand 들만 다룰 수 있다.

NE ee <
EQ ee <
GE ee <
GT ee <
LE ee <
LT ee <
GEU ee <
GTU ee <
LEU ee <
LTU ee <

비교 operation 들. ordered comparison 들은 두가지 방식이 있는데, signed 와 Unsigned 가 그것이다.

UNORDERED ee <
ORDERED ee <

추가적인 floating point unordered comparision 방식들.

UNEQ ee <
UNGE ee <
UNGT ee <
UNLE ee <
UNLT ee <

이것들은 unordered 혹은 ... 와 같다.

LTGT ee <

이것은 ordered NE 인데, 즉 !UNEQ, NaN 에 대해선 false.

SIGN_EXTEND e 1

단독 operand 를 sign-extending 한 결과를 나타낸다. operand 의 혹은 SIGN_EXTEND 표현식의 machine mode 들은 얼마나 많은 sign-extension 가 진행되었는지를 결정한다.

ZERO_EXTEND e 1

zero-extension (unsigned short 를 int 로 같이) 와 비슷하다.

TRUNCATE e 1

비슷하지만 여기서 operand 는 wider mode 를 가지고 있다.

FLOAT_EXTEND e 1 FLOAT_TRUNCATE e 1

floating-point 값들 (SFmode 를 DFmode 로 같이) 확장하는 것과 비슷하다.

FLOAT e 1

floating point 값에 대한 fixed point operand 의 변환

FIX e 1

fixed-point machine mode 를 가지는: fixed point 값으로 floating point operand 의 변환. 값은 operand 의 값이 정수일 때만 정의된다. floating-point machine mode (와 같은 mode 인 operand) 가지는: Operand 는 floating point 로 대표되는 정수 값을 생산하기 위해 zero 로 round 되어진다.

UNSIGNED_FLOAT e 1

floating point 로 unsigned fixed point operand 의 변환

UNSIGNED_FIX e 1

fixed-point machine mode 를 가지는: *unsigned* fixed point 값으로 floating point operand 의 변환 값은 operand 의 값이 정수일 때만 정의된다.

ABS e 1

Absolute value

SQRT e 1

Square root

FFS e 1

설정된 첫번째 bit 를 찾는다. 값은 arg 내 trailing zero 들의 1 + number 이며 만약 arg 가 0 일 경우 0 이다.

SIGN_EXTRACT eee b

지정된 크기와 위치의 signed bit-field 로의 참조. Operand 0 은 memory unit (보통 SImode 혹은 QImode) 인데, 이것은 field 의 첫번째 bit 를 포함한다. Operand 1 은 bit 로 나태낸 width 이다. Operand 2 는 이 field 의 첫번째 bit 앞 memory unit 내 bit 들의 number 이다. 만약 BITS_BIG_ENDIAN 가 정의되어 있다면, 첫번째 bit 는 msb 이고 operand 2 는 memory unit 의 msb 로부터 세어진다. 그렇지 않다면, 첫번째 bit 는 lsb 이고 operand 2 는 memory unit 의 lsb 로부터 세어진다.

ZERO_EXTRACT eee b

unsigned bit-field 에 대해 비슷한다.

HIGH e o

RISC machine 용. 이것들은 insn 들을 절단할 때 메모리를 아낀다.

LO_SUM ee o

LO_SUM 는 레지스터의 합계이고 상수 표현식의 low-order bit 들이다.

RANGE_INFO uuEiiiiiibbii x

Range information 를 위한 header. Operand 0 은 NOTE_INSN_RANGE_BEG insn 이다. Operand 1 는 NOTE_INSN_RANGE_END insn 이다. Operand 2 는 이 범위내에서 대체 가능한 모든 register 들의 vector 이다. Operand 3 은 범위내 call 들의 갯수이다. Operand 4 는 범위내 insn 들의 갯수. Operand 5 는 이 범위를 위한 유일한 range 번호이다. Operand 6 은 live range 의 시작 부분의 기본 block # 이다. Operand 7 은 live range 의 끝 부분의 기본 block # 이다. Operand 8 은 loop depth 이다. Operand 9 는 range 의 시작 부분에 live 한 register 들의 bitmap 이다. Operand 10 은 range 의 끝 부분에 live 한 register 들의 bitmap 이다. Operand 11 은 범위 시작 부분을 위한 marker number 이다. Operand 12 는 범위 끝 부분을 위한 marker number 이다.

RANGE_REG iiiiiiiitt x

범위내에 대체 가능할 수 있는 레지스터들. Operand 0 은 원래 pseudo register number 이다. Operand 1 은 값이 range 의 존속동안 복사되어지는 pseudo register 를 가지도록 채워진다. Operand 2 는 register 로의 범위내 reference 들의 번호 이다. Operand 3 은 범위내 집합들의 번호 혹은 레지스터의 clobber 들이다. Operand 4 는 레지스터가 가지고 있는 depth 들의 번호이다. Operand 5 는 범위의 시작 부분에서 복사본이 원래 레지스터에서 새로운 레지스터로 필요한지 안한지에 혹은 새로운 레지스터로부터의 복사본이 범위의 끝에 원래로 돌려보내야 할지 안할지에 대한 상태를 주는 copy flag 들이다. Operand 6 은 live length 이다. Operand 8 은 만약 register 가 user 변수일 경우 변수의 symbol note 이다. Operand 9 는 만약 레지스터가 user 변수일경우 변수가 내에 선언되어 있음을 가르키는 block node 이다.

RANGE_VAR eti x

Local 변수들의 ragne 들에 관한 정보. Operand 0 은 다른 range 들의 EXPR_LIST 인데, 이 범위들은 변수가 어떤 다른 pseudo register 로 복사되었는지를 나타낸다. Operand 1 은 변수가 선언되어 잇는 block 이고, Operand 2 는 distinct 범위들의 number 이다.

RANGE_LIVE bi x

현재 point 에 live 하고 있는 레지스터들에 관한 정보. Operand 0 은 live bitmap 이다. Operand 1 은 원래 block number 이다.

CONSTANT_P_RTX e x

unary '__builtin_constant_p' 표현식. 이것들은 오직 RTL 생성동안 그리고, optimize > 0 일 경우에만 방출된다. 그들은 첫번째 CSE 단계에서 제거된다.

CALL_PLACEHOLDER uuuu x

보통의 call 로 변화될 수 있는 CALL_INSN 을 위한 placeholder, 혹은 sibling (tail) call, tail recursion.

RTL 생성후 즉시, 이 placeholder 는 insn 들에 의해 call 혹은 sibcall, tail recursion 를 실행하도록 대체될 것이다.

RTX 는 4 operand 들을 가지고 있다. 첫번째 세개는 보통의 call 과 sibling call, tail recursion 로써 각자 call 을 수행하기 위한 명령어들의 list 들이다. 뒤에서 두 list 들은 아마도 NULL 일 수 있고 첫번째는 절대 NULL 이 아니다.

마지막 operand 는 tail recursion CODE_LABEL 이며 이것은 만약 잠재적인 tail recursive call 들이 발견되지 못하였을 경우 NULL 일 것이다.

tail recursion label 은 필요한데, 그걸로 우리는 우리가 call 방식을 선택한 후
LABEL_PRESERVE_P 를 깨끗히 할 수 있다.

이 tail-call elimination 의 방식은 front-end 변환들이 완료된 즉시 tree-based optimization 들로 대체되는 경향이 있다.

VEC_MERGE eee x

두 vector 값들 사이의 merge operation 을 표현한다.
Operand 0 과 1 들은 merge 되어질 vector 들이고, operand 2 는 결과의 어떤 부분이 무엇으로부터 선택되었는지를 말해주는 bitmask 이다. Set bit 들은 operand 0 을 가르키고 clear bit 들은 operand 1 을 가르킨다. 부분들은 vector 들의 mode 로 정의된다.

VEC_SELECT ee x

Vector 의 부분들을 선택하는 operation 을 설명한다. Operand 0 은 source vector 이고, operand 1 은 결과 vector 의 하위부분의 각각을 위한 CONST_INT 를 포함하는 PARALLEL 이다. 이 결과 vector 는 그것 안에 저장되어야만 하는 source subpart 의 number 를 준다.

VEC_CONCAT ee x

vector concat operation 를 설명한다. Operand 0 과 1 들은 source vector 들이고, 결과는 vector 인데, 이 vector 는 operand 0 와 1 이 결합한 형식이고 두 source vector 들의 연쇄이다.

VEC_DUPLICATE e x

작은 vector 를 input 값들을 반복함으로써 큰 것으로 변환하는 operation 을 표현한다. Output vector mode 는 input vector mode 로써 같은 submode 들을 반드시 가지고 있어야 하고 output part 들의 number 는 input part 들의 number 의 integer multiple 이여야만 한다.

SS_PLUS ee c

Signed saturation 를 가지는 더하기

US_PLUS ee c

Unsigned saturation 를 가지는 더하기

SS_MINUS ee 2

Signed saturation 를 가지는 Operand 0 빼기 operand 1.

US_MINUS ee 2

Unsigned saturation 를 가지는 Operand 0 빼기 operand 1.

SS_TRUNCATE e 1

Signed saturating truncate.

US_TRUNCATE e 1

Unsigned saturating truncate.

PHI E x

SSA phi 연산자.

인자는 2N rtx 들의 vector 이다. Element 2N+1 는 CONST_INT 인데, 이것은 element 2N 에서의 register 가 사용되었을 때 어떤 control 을 통해서 건네졌는지를 말해주는 predecessor 의 block number 를 포함한다.

참고 : PHI 는 기본 block 의 시작에서만 단지 나타난 수 있다.

제 3 절 16 주째 문서를 끝내며

다음 문서에서는 이번에 정의된 RTX 들을 C 에서 어떻게 사용하고 가공하는지 그리고 RTX 관련 함수 및 전역변수 등등을 포괄적으로 설명하도록 하겠습니다.