GCC Trees
Intermediate Representation

정원교
weongyo@hotmail.com

2003년 11월 16일

목 차

제 1 절 15 주째 강의를 시작하며

제 2 절 Machine mode 란...

Machine mode 는 machine level 에서의 data 의 크기와 format 을 지정하며 각 RTL 표현식은 machine mode 를 가지고 있습니다. Syntax tree level 에서 각 ..._TYPE 와 각 ..._DECL node 는 그 type 의 data 혹은 선언된 변수의 data 를 표현하는 machine mode 를 가지고 있습니다.

제 3 절 Machie mode 들

첫번째 인자는 C source 에서 사용되는 machine mode 의 내부 이름입니다. 변환으로 인해 UPPER_CASE 로 있지만 단어 "mode" 는 제외입니다.
두번째 인자는 RTL 과 tree 들을 출력하거나 읽어 드일때 사용되는 외부 ASCII format 형태의 machine mode 의 이름입니다. 변환으로 인해 UPPER_CASE 형태로의 이름으로 됩니다.
세번째 인자는 표현식의 종류를 기술합니다:

MODE_INT - 정수
MODE_FLOAT - 실수
MODE_PARTIAL_INT - PQImode 와 PHImode, PSImode, PDImode
MODE_CC - 레지스터내에서의 Condition Code 를 표현할 때 사용되는 mode 들
MODE_COMPLEX_INT, MODE_COMPLEX_FLOAT - 복소수
MODE_VECTOR_INT, MODE_VECTOR_FLOAT - vector
MODE_RANDOM - 다른 기타 등등

네번째 인자는 비트로 나타낸 object 의 상대적 크기이며 이렇게 해서 우리는 1 바이트보다 작은 mode 들을 가질 수 있습니다.
다섯번째 인자는 바이트로 나타낸 object 의 상대적 크기입니다. 만약 size 를 나타내는 것이 의미없거나 아직 결정되지 않았다면 0 으로 합니다. 한 바이트의 크기는 tm.h 내의 BITS_PER_UNIT 에 의해 결정됩니다.
여섯번째 인자는 object 의 subunit 들의 상대적인 크기입니다. 이것은 complex 들과 vector 들을 제외하고는 다섯번째 인자와 같습니다. 이것은 complex 들과 vector 들이 실제로는 같은 크기의 많은 subunit 들로 구성되어 있기 때문입니다.
일곱번째 인자는 같은 class 에서의 좀 더 넓은 mode 를 가르킵니다. 만약 존재하지 않는다면 0 입니다. Vector mode 들은 이 field 를 다음 vector size 를 가르키는데 사용하며 그렇게 함으로써 우리는 다른 vector 의 mode 들을 통해 반복할 수 있습니다. 순서는 byte 크기가 증가하는 순으로 됩니다. HI 전에 QI 가 오며 SI 전에 HI 가 오는 형식입니다.
여덟번째 인자는 vector 혹은 complex 내의 internal element 들의 mode 이며 적용될 수 없는 것일 경우 VOIDmode 를 갖습니다.

*1	*2	*3	*4	*5	*6	*7	*8
VOIDmode	VOID	MODE_RANDOM	0	0	0	VOIDmode	VOIDmode
BImode	BI	MODE_INT	1	1	1	QImode	VOIDmode
QImode	QI	MODE_INT	BPU	1	1	HImode	VOIDmode
HImode	HI	MODE_INT	BPU* 2	2	2	SImode	VOIDmode
SImode	SI	MODE_INT	BPU* 4	4	4	DImode	VOIDmode
DImode	DI	MODE_INT	BPU* 8	8	8	TImode	VOIDmode
TImode	TI	MODE_INT	BPU*16	16	16	OImode	VOIDmode
OImode	OI	MODE_INT	BPU*32	32	32	VOIDmode	VOIDmode
PQImode	PQI	MODE_PARTIAL_INT	BPU	1	1	PHImode	VOIDmode
PHImode	PHI	MODE_PARTIAL_INT	BPU* 2	2	2	PSImode	VOIDmode
PSImode	PSI	MODE_PARTIAL_INT	BPU* 4	4	4	PDImode	VOIDmode
PDImode	PDI	MODE_PARTIAL_INT	BPU* 8	8	8	VOIDmode	VOIDmode
QFmode	QF	MODE_FLOAT	BPU	1	1	HFmode	VOIDmode
HFmode	HF	MODE_FLOAT	BPU* 2	2	2	TQFmode	VOIDmode
TQFmode	TQF	MODE_FLOAT	BPU* 3	3	3	SFmode	VOIDmode
SFmode	SF	MODE_FLOAT	BPU* 4	4	4	DFmode	VOIDmode
DFmode	DF	MODE_FLOAT	BPU* 8	8	8	XFmode	VOIDmode
XFmode	XF	MODE_FLOAT	BPU*12	12	12	TFmode	VOIDmode
TFmode	TF	MODE_FLOAT	BPU*16	16	16	VOIDmode	VOIDmode
QCmode	QC	MODE_COMPLEX_FLOAT	BPU* 2	2	1	HCmode	QFmode
HCmode	HC	MODE_COMPLEX_FLOAT	BPU* 4	4	2	SCmode	HFmode
SCmode	SC	MODE_COMPLEX_FLOAT	BPU* 8	8	4	DCmode	SFmode
DCmode	DC	MODE_COMPLEX_FLOAT	BPU*16	16	8	XCmode	DFmode
XCmode	XC	MODE_COMPLEX_FLOAT	BPU*24	24	12	TCmode	XFmode
TCmode	TC	MODE_COMPLEX_FLOAT	BPU*32	32	16	VOIDmode	TFmode
CQImode	CQI	MODE_COMPLEX_INT	BPU* 2	2	1	CHImode	QImode
CHImode	CHI	MODE_COMPLEX_INT	BPU* 4	4	2	CSImode	HImode
CSImode	CSI	MODE_COMPLEX_INT	BPU* 8	8	4	CDImode	SImode
CDImode	CDI	MODE_COMPLEX_INT	BPU*16	16	8	CTImode	DImode
CTImode	CTI	MODE_COMPLEX_INT	BPU*32	32	16	COImode	TImode
COImode	COI	MODE_COMPLEX_INT	BPU*64	64	32	VOIDmode	OImode
V2QImode	V2QI	MODE_VECTOR_INT	BPU* 2	2	1	V4QImode	QImode
V2HImode	V2HI	MODE_VECTOR_INT	BPU* 4	4	2	V8QImode	HImode
V2SImode	V2SI	MODE_VECTOR_INT	BPU* 8	8	4	V16QImode	SImode
V2DImode	V2DI	MODE_VECTOR_INT	BPU*16	16	8	V8SImode	DImode
V4QImode	V4QI	MODE_VECTOR_INT	BPU* 4	4	1	V2HImode	QImode
V4HImode	V4HI	MODE_VECTOR_INT	BPU* 8	8	2	V2SImode	HImode
V4SImode	V4SI	MODE_VECTOR_INT	BPU*16	16	4	V2DImode	SImode
V4DImode	V4DI	MODE_VECTOR_INT	BPU*32	32	8	V8DImode	DImode
V8QImode	V8QI	MODE_VECTOR_INT	BPU* 8	8	1	V4HImode	QImode

V8HImode	V8HI	MODE_VECTOR_INT	BPU*16	16	2	V4SImode	HImode
V8SImode	V8SI	MODE_VECTOR_INT	BPU*32	32	4	V4DImode	SImode
V8DImode	V8DI	MODE_VECTOR_INT	BPU*64	64	8	VOIDmode	DImode
V16QImode	V16QI	MODE_VECTOR_INT	BPU*16	16	1	V8HImode	QImode
V2SFmode	V2SF	MODE_VECTOR_FLOAT	BPU* 8	8	4	V4SFmode	SFmode
V2DFmode	V2DF	MODE_VECTOR_FLOAT	BPU*16	16	8	V8SFmode	DFmode
V4SFmode	V4SF	MODE_VECTOR_FLOAT	BPU*16	16	4	V2DFmode	SFmode
V4DFmode	V4DF	MODE_VECTOR_FLOAT	BPU*32	32	8	V8DFmode	DFmode
V8SFmode	V8SF	MODE_VECTOR_FLOAT	BPU*32	32	4	V4DFmode	SFmode
V8DFmode	V8DF	MODE_VECTOR_FLOAT	BPU*64	64	8	VOIDmode	DFmode
V16SFmode	V16SF	MODE_VECTOR_FLOAT	512	64	4	VOIDmode	SFmode
BLKmode	BLK	MODE_RANDOM	0	0	0	VOIDmode	VOIDmode
CCmode	CC	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
아래는 추가적인 Machine mode 들이다. 여기서 추가된 6 개의 요소는 i386 용을 위한 것이다.							
CCGCmode	CCGC	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
CCGOCmode	CCGOC	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
CCNOmode	CCNO	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
CCZmode	CCZ	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
CCFPmode	CCFP	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode
CCFPUmode	CCFPU	MODE_CC	BPU* 4	4	4	VOIDmode	VOIDmode

VOIDmode

VOIDmode 는 CONST_INT RTL 표현식과 같이 mode 를 지정할 필요가 없을 때 사용됩니다.

PQImode, PHImode, PSImode, PDImode

몇몇 machine 들상에서 pointer 들은 int 들과 그것들을 구분하기 위해서 이 type 들을 사용한다. 만약 pointer 가 4 바이트이지만, 충분하지 않은 bit 들을 가지고 있다면 유용하다. 그것은 정수처럼 넓지 않다.

QCmode, HCmode, SCmode, DCmode, XCmode, TCmode

복소수 (Complex) mode 들.

V2QImode, V2HImode, V2SImode, V2DImode, V4QImode, V4HImode, V4SImode, V4DImode, V8QImode, V8HImode, V8SImode, V8DImode, V16QImode, V2SFmode, V2DFmode, V4SFmode, V4DFmode, V8SFmode, V8DFmode, V16SFmode

Vector mode 들. V1xx vector mode 들은 존재하지 않으며 이것들은 보통의 scalar mode 와 동일하다. vector 들을 위한 좀 더 넓은 mode field 는 HI 전에 QI 가 오는 bit size 의 증가 순을 따르며, SI 앞에는 HI, 같은 bit size 내에서는 DI 앞에 SI 가 따른다.

CCGCmode, CCGOCmode, CCNOmode, CCZmode, CCFPmode, CCFPUmode

Condition code 를 표현하는데 필요한 다른 여분의 mode 들을 추가한다.

i386 용으로 우리는 floating-point equality comparison 가 수행될 때 분리된 mode 들이 필요하다.

Overflow flag 가 설정되지 않은 상태를 요구하는 0 에 대한 비교를 가르키기 위해 CCNO 를 더하였다. Sign bit test 가 대신 사용되며 test 들의 “a&b>0” type 을 형성하는데 사용될 수 있다.

Carry flag 내에서 지정되지 않은 garbage 를 허락하는 0 에 대한 비교를 가르키기 위해 CCGC 를 더하였다. 이 mode 는 inc/dec 명령어들에 의해 사용된다.

Carry 와 Overflow flag 내에서 지정되지 않은 garbage 를 허락하는 0 에 대한 비교를 가르키기 위해 CCGOC 를 더하였다. 이 mode 는 sub/cmp/add operation 들을 사용하는 0 에 대해 (a-b) 와 (a+b) 의 비교를 실험하는데 사용된다.

단지 Zero flag 가 valid 임을 나타내기 위해 CCZ 를 추가.

제 4 절 Machine mode 들을 사용하기 위한 선언들

앞 절에서 GCC 내부에서 사용하는 machine mode 들에 대해서 알아 봤으므로, 이를 실제로 사용하는 부분에 대해서 알아 보자.

모든 machine mode 들을 가지고 있는 enum class 를 만는데, 아래와 같이 선언된다. Machine mode 의 갯수를 잘 알수 있도록 끝에 MAX_MACHINE_MODE 를 추가하였다.

```
enum machine_mode {
  VOIDmode,
  BImode, QImode, HImode, SImode, DImode, TImode, OImode,
  PQImode, PHImode, PSImode, PDImode,
  QFmode, HFmode, TQFmode, SFmode, DFmode, XFmode, TFmode,
  QCmode, HCmode, SCmode, DCmode, XCmode, TCmode,
  CQImode, CHImode, CSImode, CDImode, CTImode, COImode,
  V2QImode, V2HImode, V2SImode, V2DImode,
  V4QImode, V4HImode, V4SImode, V4DImode, V8QImode, V8HImode, V8SImode, V8DImode,
  V16QImode,
  V2SFmode, V2DFmode,
  V4SFmode, V4DFmode,
  V8SFmode, V8DFmode, V16SFmode,
  BLKmode,
  CCmode,
  CCGCmode, CCGOCmode, CCNOmode, CCZmode, CCFPmode, CCFPUmode,
  MAX_MACHINE_MODE
};
```

문자열 형태로의 mode MODE 의 이름을 얻을 때 사용된다. 이름에 대한 접근은 아래에 같이 선언되어 있는 GET_MODE_NAME () 매크로를 통해서 이루어지게 된다.

```
#define GET_MODE_NAME(MODE)             (mode_name[(int) (MODE)])

const char * const mode_name[(int) MAX_MACHINE_MODE] =
{
  "VOID",
  "BI", "QI", "HI", "SI", "DI", "TI", "OI",
  "PQI", "PHI", "PSI", "PDI",
  "QF", "HF", "TQF", "SF", "DF", "XF", "TF",
  "QC", "HC", "SC", "DC", "XC", "TC",
  "CQI", "CHI", "CSI", "CDI", "CTI", "COI",
  "V2QI", "V2HI", "V2SI", "V2DI",
  "V4QI", "V4HI", "V4SI", "V4DI",
  "V8QI", "V8HI", "V8SI", "V8DI",
  "V16QI",
  "V2SF", "V2DF",
  "V4SF", "V4DF",
  "V8SF", "V8DF", "V16SF",
  "BLK",
  "CC",
  "CCGC", "CCGOC", "CCNO", "CCZ", "CCFP", "CCFPU",
};
```

mode MODE 를 표현하는 object 의 일반 분류를 얻는데 사용된다. enum mode_class 에는 machine mode 들이 속해 있는 class 의 종류를 말해주고 있으며, mode_class 전역 변수의 경우 각 machine mode 의

class 를 나타내고 있다. 각 모드에 대한 설명은 위 절에서 해 놓았으므로 그것을 참고하고, 각 모드의 class 를 획득 하기 위한 매크로는 아래에 선언되어 있는 GET_MODE_CLASS 매크로를 이용하면 된다.

```
#define GET_MODE_CLASS(MODE)            (mode_class[(int) (MODE)])

enum mode_class {
  MODE_RANDOM, MODE_INT, MODE_FLOAT, MODE_PARTIAL_INT, MODE_CC,
  MODE_COMPLEX_INT, MODE_COMPLEX_FLOAT,
  MODE_VECTOR_INT, MODE_VECTOR_FLOAT,
  MAX_MODE_CLASS
};

const enum mode_class mode_class[(int) MAX_MACHINE_MODE] =
{
  MODE_RANDOM,
  MODE_INT, MODE_INT, MODE_INT, MODE_INT, MODE_INT, MODE_INT, MODE_INT,
  MODE_PARTIAL_INT, MODE_PARTIAL_INT, MODE_PARTIAL_INT, MODE_PARTIAL_INT,
  MODE_FLOAT, MODE_FLOAT, MODE_FLOAT, MODE_FLOAT, MODE_FLOAT, MODE_FLOAT,
    MODE_FLOAT,
  MODE_COMPLEX_FLOAT, MODE_COMPLEX_FLOAT, MODE_COMPLEX_FLOAT, MODE_COMPLEX_FLOAT,
    MODE_COMPLEX_FLOAT, MODE_COMPLEX_FLOAT,
  MODE_COMPLEX_INT, MODE_COMPLEX_INT, MODE_COMPLEX_INT, MODE_COMPLEX_INT,
    MODE_COMPLEX_INT, MODE_COMPLEX_INT,
  MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT,
  MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT,
  MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT, MODE_VECTOR_INT,
  MODE_VECTOR_INT,
  MODE_VECTOR_FLOAT, MODE_VECTOR_FLOAT,
  MODE_VECTOR_FLOAT, MODE_VECTOR_FLOAT,
  MODE_VECTOR_FLOAT, MODE_VECTOR_FLOAT, MODE_VECTOR_FLOAT,
  MODE_RANDOM,
  MODE_CC,
  MODE_CC, MODE_CC, MODE_CC, MODE_CC, MODE_CC, MODE_CC,
};
```

또한 각 mode 를 위한 class 마다 공통되는 class 가 존재하는데, 정수, 실수, 복소수, vector 등등으로 크게 생각될 수 있다. 이와 같이 큰 덩어리로 생각될 필요가 있을 경우를 대비하여, ...MODE_P 매크로가 몇 개 존재하는데, 그것은 다은과 같다.

만약 MODE 가 integral mode 라면 0 이 아닌 값을 반환

```
#define INTEGRAL_MODE_P(MODE)                   \
  (GET_MODE_CLASS (MODE) == MODE_INT            \
   || GET_MODE_CLASS (MODE) == MODE_PARTIAL_INT \
   || GET_MODE_CLASS (MODE) == MODE_COMPLEX_INT \
   || GET_MODE_CLASS (MODE) == MODE_VECTOR_INT)
```

만약 MODE 가 부동-소수점 mode 라면 0 이 아닌 값을 반환

```
#define FLOAT_MODE_P(MODE)              \
  (GET_MODE_CLASS (MODE) == MODE_FLOAT  \
   || GET_MODE_CLASS (MODE) == MODE_COMPLEX_FLOAT \
   || GET_MODE_CLASS (MODE) == MODE_VECTOR_FLOAT)
```

만약 MODE 가 complex mode 이라면 0 이 아닌 값을 반환

```
#define COMPLEX_MODE_P(MODE)                    \
  (GET_MODE_CLASS (MODE) == MODE_COMPLEX_INT    \
   || GET_MODE_CLASS (MODE) == MODE_COMPLEX_FLOAT)
```

만약 MODE 가 vector mode 라면 0 이 아닌 값을 반환

```
#define VECTOR_MODE_P(MODE)                     \
  (GET_MODE_CLASS (MODE) == MODE_VECTOR_INT     \
   || GET_MODE_CLASS (MODE) == MODE_VECTOR_FLOAT)
```

mode MODE 의 object 의 비트 크기를 얻는데 사용된다. GET_MODE_BITSIZE 매크로를 이용하여 각 mode 의 크기를 얻는데 접근한다.

```
#define GET_MODE_BITSIZE(MODE)  (mode_bitsize[(int) (MODE)])

const unsigned short mode_bitsize[(int) MAX_MACHINE_MODE] =
{
  0,
  1, 8, 8*2, 8*4, 8*8, 8*16, 8*32,
  8, 8*2, 8*4, 8*8,
  8, 8*2, 8*3, 8*4, 8*8, 8*12, 8*16,
  8*2, 8*4, 8*8, 8*16, 8*24, 8*32,
  8*2, 8*4, 8*8, 8*16, 8*32, 8*64,
  8*2, 8*4, 8*8, 8*16,
  8*4, 8*8, 8*16, 8*32,
  8*8, 8*16, 8*32, 8*64,
  8*16,
  8*8, 8*16,
  8*16, 8*32,
  8*32, 8*64, 512,
  0,
  8*4,
  8*4, 8*4, 8*4, 8*4, 8*4, 8*4,
};
```

mode MODE 의 object 의 바이트 크기를 얻는데 사용되며, GET_MODE_SIZE 매크로를 이용하여 각 mode 의 크기를 얻는데 접근한다.

```
#define GET_MODE_SIZE(MODE)             (mode_size[(int) (MODE)])

const unsigned char mode_size[(int) MAX_MACHINE_MODE] =
{
  0,
  1, 1, 2, 4, 8, 16, 32,
  1, 2, 4, 8,
  1, 2, 3, 4, 8, 12, 16,
  2, 4, 8, 16, 24, 32,
  2, 4, 8, 16, 32, 64,
  2, 4, 8, 16,
  4, 8, 16, 32,
  8, 16, 32, 64,
  16,
  8, 16,
  16, 32,
```

```
  32, 64, 64,
  0,
  4,
  4, 4, 4, 4, 4, 4,
};
```

mode MODE 의 object 의 basic part 들의 바이트 크기를 얻는데 사용되며, GET_MODE_UNIT_SIZE 매크로를 이용하여 각 mode 의 값에 접근하게 된다. 그리고 Object 내의 unit 들의 갯수를 얻는데는 GET_MODE_NUNITS 매크로를 이용한다.

```
#define GET_MODE_UNIT_SIZE(MODE)        (mode_unit_size[(int) (MODE)])

const unsigned char mode_unit_size[(int) MAX_MACHINE_MODE] =
{
  0,
  1, 1, 2, 4, 8, 16, 32,
  1, 2, 4, 8,
  1, 2, 3, 4, 8, 12, 16,
  1, 2, 4, 8, 12, 16,
  1, 2, 4, 8, 16, 32,
  1, 2, 4, 8,
  1, 2, 4, 8,
  1, 2, 4, 8,
  1,
  4, 8,
  4, 8,
  4, 8, 4,
  0,
  4,
  4, 4, 4, 4, 4, 4,
};

#define GET_MODE_NUNITS(MODE)  \
  ((GET_MODE_UNIT_SIZE ((MODE)) == 0) ? 0 \
   : (GET_MODE_SIZE ((MODE)) / GET_MODE_UNIT_SIZE ((MODE))))
```

해당 machine mode 보다 좀 더 넓은 mode 를 얻는데 사용된다. 예를 들면 QI → HI → SI → DI → TI 순으로 넓다. 그리고 이 전역변수에 접근하기 위한 매크로는 GET_MODE_WIDER_MODE 를 사용한다.

```
const unsigned char mode_wider_mode[(int) MAX_MACHINE_MODE] =
{
  (unsigned char) VOIDmode,
  (unsigned char) QImode, (unsigned char) HImode, (unsigned char) SImode,
    (unsigned char) DImode, (unsigned char) TImode, (unsigned char) OImode,
    (unsigned char) VOIDmode,
  (unsigned char) PHImode, (unsigned char) PSImode, (unsigned char) PDImode,
    (unsigned char) VOIDmode,
  (unsigned char) HFmode, (unsigned char) TQFmode, (unsigned char) SFmode,
    (unsigned char) DFmode, (unsigned char) XFmode, (unsigned char) TFmode,
    (unsigned char) VOIDmode,
  (unsigned char) HCmode, (unsigned char) SCmode, (unsigned char) DCmode,
    (unsigned char) XCmode, (unsigned char) TCmode, (unsigned char) VOIDmode,
  (unsigned char) CHImode, (unsigned char) CSImode, (unsigned char) CDImode,
    (unsigned char) CTImode, (unsigned char) COImode, (unsigned char) VOIDmode,
```

```
  (unsigned char) V4QImode, (unsigned char) V8QImode, (unsigned char) V16QImode,
    (unsigned char) V8SImode,
  (unsigned char) V2HImode, (unsigned char) V2SImode, (unsigned char) V2DImode,
    (unsigned char) V8DImode,
  (unsigned char) V4HImode, (unsigned char) V4SImode, (unsigned char) V4DImode,
    (unsigned char) VOIDmode,
  (unsigned char) V8HImode,
  (unsigned char) V4SFmode, (unsigned char) V8SFmode,
  (unsigned char) V2DFmode, (unsigned char) V8DFmode,
  (unsigned char) V4DFmode, (unsigned char) VOIDmode, (unsigned char) VOIDmode,
  (unsigned char) VOIDmode,
  (unsigned char) VOIDmode,
  (unsigned char) VOIDmode, (unsigned char) VOIDmode, (unsigned char) VOIDmode,
    (unsigned char) VOIDmode, (unsigned char) VOIDmode, (unsigned char) VOIDmode,
};
```

Word 내의 모든 bit 가 1 을 포함하는 bitmask 를 얻는데, 이 word 는 mode MODE 내부에 맞춘 것이다. 이 전역변수에 접근하기 위해서는 GET_MODE_MASK 매크로를 사용한다.

```
#define GET_MODE_MASK(MODE) mode_mask_array[(int) (MODE)]

const unsigned int mode_mask_array[(int) MAX_MACHINE_MODE] =
{
  ((0) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (0)) - 1,

  ((1) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (1)) - 1,
  ((8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8)) - 1,
  ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
  ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
  ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
  ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
  ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,

  ((8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8)) - 1,
  ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
  ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
  ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,

  ((8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8)) - 1,
  ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
  ((8*3) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*3)) - 1,
  ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
  ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
  ((8*12) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*12)) - 1,
  ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,

  ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
  ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
  ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
  ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
  ((8*24) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*24)) - 1,
  ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,
```

```
    ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
    ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,
    ((8*64) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*64)) - 1,

    ((8*2) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*2)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,

    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
    ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,

    ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
    ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,
    ((8*64) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*64)) - 1,

    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,

    ((8*8) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*8)) - 1,
    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,

    ((8*16) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*16)) - 1,
    ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,

    ((8*32) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*32)) - 1,
    ((8*64) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*64)) - 1,
    ((512) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (512)) - 1,

    ((0) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (0)) - 1,

    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,

    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
    ((8*4) >= (8 * 4)) ? ~(unsigned int) 0 : ((unsigned int) 1 << (8*4)) - 1,
};
```

Vector 내의 inner element 들의 mode 를 반환한다. GET_MODE_INNER 매크로를 이용하여 접근 가능하다.

```
#define GET_MODE_INNER(MODE) inner_mode_array[(int) (MODE)]

const enum machine_mode inner_mode_array[(int) MAX_MACHINE_MODE] =
{
```

```
  VOIDmode,
  VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode,
  VOIDmode, VOIDmode, VOIDmode, VOIDmode,
  VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode,
  QFmode, HFmode, SFmode, DFmode, XFmode, TFmode,
  QImode, HImode, SImode, DImode, TImode, OImode,
  QImode, HImode, SImode, DImode,
  QImode, HImode, SImode, DImode, QImode, HImode, SImode, DImode,
  QImode,
  SFmode, DFmode,
  SFmode, DFmode,
  SFmode, DFmode, SFmode,
  VOIDmode,
  VOIDmode,
  VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode, VOIDmode,
};
```

각 class 에 대해 그 class 안의 narrowest mode 를 얻는데 사용되며, GET_CLASS_NARROWEST_MODE 매크로를 이용하여 접근이 가능하다.

```
#define GET_CLASS_NARROWEST_MODE(CLASS) class_narrowest_mode[(int) (CLASS)]

const enum machine_mode class_narrowest_mode[(int) MAX_MODE_CLASS] =
{
  VOIDmode,
  QImode,
  QFmode,
  PQImode,
  CCmode,
  CQImode,
  QCmode,
  V2QImode,
  V2SFmode
};
```

마지막 전역변수에 대해서 살펴보면, $prefix/gcc/emit-rtl.c 파일에 선언되어 있으며 Integer mode 들을 정의한다. integer mode 들의 size 들은 BITS_PER_UNIT 와 BITS_PER_WORD 이고, mode 의 class 는 Pmode 이고 mode 의 size 는 POINTER_SIZE 이다.

설정되는 값들을 살펴보면, byte_mode 는 BImode 로 설정이 된고, word_mode 는 SImode, double_mode 는 DFmode, ptr_mode 는 SImode 가 각각 설정된다.

```
enum machine_mode byte_mode;
enum machine_mode word_mode;
enum machine_mode double_mode;
enum machine_mode ptr_mode;
```

제 5 절 Machine mode 를 위한 함수들

```
enum machine_mode mode_for_size PARAMS ((unsigned int,
                                         enum mode_class, int));
```

주어진 size SIZE 와 mode class CLASS 의 data 를 위한 mode 를 반환한다. 만약 LIMIT 가 0 이 아니라면, MAX_FIXED_MODE_SIZE 보다 큰 mode 들을 사용하지 않는다. 만약 다른 쓸만한 mode 를 찾지 못할 경우 값은 BLKmode 이다.

```
enum machine_mode smallest_mode_for_size
                                PARAMS ((unsigned int, enum mode_class));
```

비슷하지만, 주어진 넓이에 맞는 가장 작은 mode 를 찾는다.

```
enum machine_mode int_mode_for_mode PARAMS ((enum machine_mode));
```

Input mode 와 정확히 같은 크기의 정수 mode 를 반환하며, 실패시 BLKmode 를 반환.

```
enum machine_mode get_best_mode PARAMS ((int, int, unsigned int,
                                         enum machine_mode, int));
```

Bit field 에 접근하는데 사용할 최고의 mode 를 찾는다.

```
unsigned get_mode_alignment PARAMS ((enum machine_mode));
```

Alignment 를 결정한다, 1<=result<=BIGGEST_ALIGNMENT.