Linux için Satıriçi Sembolik Makina Dili (Inline Assembly)

Parçaları bir araya getirelim

Yazan:
Bharata B. Rao
<rbharata (at) in.ibm.com>

Çeviren:
Ramazan Yağmur
<ryagmur (at) lycos.co.uk>

Mart 2001

Özet

Bharata B. Rao Linux platformunda x86 için Sembolik Makina Dilinin (Assembly) genel kullanımı ve yapısı hakkındaki bu rehberinde Satıriçi Sembolik Makina Dilinin (Inline Assembly) temelini ve çeşitli kulanımlarını kapsıyor, bazı temel Satıriçi Sembolik Makina Dili kod örnekleri veriyor ve Linux çekirdeğindeki bazı Satıriçi Sembolik Makina Dili kodları açıklıyor.

Konu Başlıkları

Sürüm Bilgileri

IBM Linux Teknoloji Merkezi , IBM Yazılım Lab. Hindistan

Geçmiş

1.0	Mart2001	Çeviri: Ramazan Yağmur, Özgün Belge: Bharata B. Rao
İlk sürüm		

Yasal Açıklamalar

1. Giriş

Eğer Linux çekirdek geliştiricisi iseniz, muhtemelen yüksek mimariye bağlı kodlar yazıyor veya bu kodları düzenliyorsunuzdur. Büyük olasılıkla bunu da Sembolik Makina Dili (Assembly) kodlarını, C kodlarının arasına yerleştirerek (Inline Assembly diye bilenen yöntemle) yapıyorsunuzdur. Şimdi Satıriçi sembolik makina dilinin Linux'daki özel kullanımına bakalım. (Çalışmalarımızı IA32 Assembly ile sınırlandıracağız.)

2. Kısaca GNU Assembler sözdizimi

Önce Linux GCC'de temel sembolik makina dili sözdizimine bir bakalım. Linux için GNU C Derleyicisi AT&T sembolik makina dili sözdizimini kullanıyor. Sözdiziminin temel kuralları aşağıda listelenmiştir. (Listenin tamamını içermiyor sadece Satıriçi Sembolik Makina Diline bağlı olanları içeriyor.)

Yazmaç adlandırma

Yazmaç (register) isimleri `%` öneki ile kullanılır. Şöyleki `eax` kullanılacaksa `%eax` olarak kullanılmalı.

Kaynak ve hedef sıralaması

Her komutta kaynak önce gelir ve hedef onu takip eder. Bu Intel söz diziminde olan önce hedef sonra kaynak sıralaması ile arasında olan farktır.

```
mov %eax, %ebx -- eax'teki değeri ebx'e taşır.
```

Terim büyüklüğü

Komutlar terimin (operand) `byte`, `word` veya `long` olmasına bağlı olarak `b`, `w` veya `l` sonekleri ile kullanılır. Bu mecburi değildir. GCC terimin değerini okuyarak uygun soneki ekler. Ama soneki kendiniz yazmanız kodun okunurluğunu arttırır ve olası yanlış tahminleri engeller.

```
movb %al,  %bl   -- Byte taşıma
movw %ax,  %bx   -- Word taşıma
movl %eax, %ebx  -- Longword taşıma
```

Sabit terimler

Sabit terimler `$` işareti ile belirlenir.

```
movl $0xffff, %eax -- 0xffff 'ın değerini eax yazmacına taşır.
```

Dolaylı bellek yönlendirmeleri

Belleğe dolaylı yönlendirmeler `()` kullanılarak yapılır.

```
movb (%esi), %al -- esi tarafından belirtilen baytı bellekten
                                    al yazmacına taşır.
```

3. Satıriçi Sembolik Makina Dili

GCC Satıriçi Sembolik Makina Dili için aşağıda gösterilen özel "asm" yapısını kullanır:

GCC'nin "asm" yapısı

```
    asm (
        assembler şablonu
        : çıktı terimleri                       (isteğe bağlı)
        : girdi terimleri                       (isteğe bağlı)
        : geri dönen yazmaçlar                  (isteğe bağlı)
        );
```

Bu örnekte sembolik makina dili şablonu sembolik makina dili komutlarından ibarettir. Girdi terimleri, girdi terimleri gibi çalışan C deyimleridir. Çıktı terimleri sembolik makina dili komutlarının çıktılarının uygulanacağı C deyimleridir.

Satıriçi Sembolik Makina Dili yönetim yeteneği ve çıktılarını C değişkenleri ile görülebilir kılması ile birinci dereceden önem kazanır. Yeteneği sayesinde "asm" sembolik makina dili komutları ile onları içeren C programı arasında bir arayüz niteliğindedir.

Aralarındaki temel ve önemli fark Temel Satıriçi Sembolik Makina Dili komutlardan, İleri Sembolik Makina Dili terimlerden ibarettir. Örneklendirme için aşağıdaki örneğe bakın:

Örnek 1. Satıriçi Sembolik Makina Dili temel yapısı

```
{
        int a=10, b;
        asm ("movl %1, %%eax;
              movl %%eax, %0;"
                :"=r"(b)        /* çıktı */
                :"r"(a)         /* girdi */
                :"%eax");       /* geri dönen yazmaç */
}
```

Örneğimizde Sembolik Makina Dili komutları ile `b`'nin değerini `a`'ya eşitledik. Aşağıdaki noktaları not edin.

- `b` çıktı terimidir, `%0` ile gösterilmiştir ve `a` girdi terimidir, `%1` işe gösterilmiştir.
- `r`, `a` ve `b`'nin yazmaçlarda kalmasını sağlayan belirteçtir. Şu unutulmamalıdır ki çıktı teriminin belirtecinin her zaman `=` ile çıktı terimi olduğu belirtilmelidir.
- "asm" yapısı içinde `%eax` yazmacını kullanmak için bir tane daha `%` kullanılmalıdır, diğer bir deyişle `%%eax`, "asm" `%0 %1` vb. değişkenleri tanımlamak için kullanır. Tek `%`'li herşey girdi/çıktı terimi olarak muamele görür.
- Geri dönen yazmaç, `%eax`, üçüncü iki noktanın sonunda GCC'ye `%eax`'in "asm" içinde değiştirildiğini söyler, bundan dolayı GCC diğer değerleri tutmak için `%eax`'i kullanmayacaktır.
- `movl %1, %%eax`, `a`'nın değerini `%eax`'e taşır ve `movl %%eax,%0`, `%eax`'in değerini `b`'ye taşır.
- "asm" yapısının çalıştırılması bitince `b` çıktı terimi, güncellenmiş değeri yansıtacaktır. Yani `b`'nin "asm" içindeki değişikliklerinin "asm" dışına da yansıması beklenir.

Şimdi bu maddeleri biraz daha ayrıntılı olarak inceleyelim.

4. Sembolik Makina Dili Şablonu

Sembolik Makina Dili şablonu C programı içine yerleştirilmiş bir takım Sembolik Makina Dili komutlarıdır. (Tek bir komut ya da bir grup komut). Ya her komut çift tırnak içine yazılmalı ya da grubun tamamı çift tırnak içinde olmalıdır. Her komut bir sonlandırıcı ile sonlandırılmalıdır. Geçerli sonlandırıcılar satırsonu (`\n`) ve noktalı virgüldür (`;`). GCC'nin oluşturacağı Sembolik Makina Dili dosyasının okunurluğunu arttırmak için satırsonundan (`\n`) sonra düzenleyici olarak sekme (`\t`) kullanılmalıdır. Komutlar, terim olarak gösterilmiş C deyimlerini numaralarla gösterirler. `%0 %1` gibi.

Eğer derleyicinin "asm" içindeki kodları düzenlemediğinden eminseniz, "asm"den sonra "volatile" kullanabilirsiniz. Eğer programınız ANSI C uyumlu olmalıysa `volatile` yerine `__asm__and__volatile__` kullanmalısınız.

5. Terimler

C deyimleri "asm" içindeki Sembolik Makina Dili komutları için terim gibi çalışırlar. Terimler Satıriçi Sembolik Makina Dilinin temel öğeleridir. Çünkü, C programının deyimleriyle iyi işler başarır.

Her terim, terim belirteci ile gösterilir ve parantez içinde gösterilen C deyimi onu takip eder.

Örneğin: "belirteç" (C deyimi). Terim belirtecinin en önemli görevi terimin kipini bulmaktır.

Hem girdi hem de çıktı bölümünde birden fazla terim kullanabilirsiniz. Her terim virgülle ayrılarak yazılır.

Terimler Sembolik Makina Dili şablonunda numaralarla gösterilirler. Eğer toplam `n` adet terim varsa (hem girdi hem de çıktı bölümleri dahil), ilk terim `0` olarak numaralandırılır ve bu düzgün artarak devam eder ve son girdi terim `n-1` olarak numaralandırılır. Terimlerin toplam sayısı 10 veya makine tanımındaki komut düzenindeki maksimum terime sahip olan kadardır.

6. Geri Dönen Yazmaç Listesi

Eğer "asm" yapısındaki komutlar donanım yazmaçlarını kullanıyorsa , GCC'ye yazmaçları kendimiz kullanıp kendimiz değiştireceğimizi söyleyebiliriz. GCC sonuç olarak bu yazmaçlara atanan değerlerin kesin olarak geçerli olacağını düşünmez. Geri dönen yazmaçları girdi veya çıktıya yazmak gerekli değildir, çünkü GCC "asm"nin onları kullandığını bilir. (Çünkü açık belirteç olarak tanımlanmışlardır). Eğer komutlar başka yazmaçları da açık veya gizli olarak kullanıyorsa (ve bu yazmaçlar girdi ve çıktı bölümlerinin birinde verilmemiş ise) geri dönen yazmaçlar listesinde bulunmak zorundadır. Geri dönen yazmaçlar üçüncü iki noktadan sonra bir dizi olarak belirtilir.

Eğer komutlar belleği daha önceden belirtilmemiş bir biçimde veya kapalı olarak değiştiriyorsa geri dönen yazmaçlar bölümüne "memory" ayrılmış kelimesi yazılmalıdır. Bu GCC'ye komutlar sırasında önbelleklenmiş yazmaçları hafızada tutmamasını söyler.

7. Terim Belirteçleri

Daha önce anlatıldığı gibi, "asm" yapısındaki her terim parantez içindeki bir C deyimi tarafından takip edilen bir terim belirteci tarafından dizi olarak açıklanmalıdır. Terim belirteçleri gerektiği gibi terimin komuttaki adresleme yöntemini tespit ederler. Bunun dışında belirteçler:

- Terimin bir yazmaçda tutulabilirliğini ve tutulabileceği yazmaç türünü
- Terimin bellek göstericisi mi olduğunu ve bu durumda tutabileceği adres türlerini
- Terimin direk bir sabit mi olduğunu da belirtir.

Belirteçler karşılaştırma için iki terim gerektirebilirler. Geçerli bir kaç belirteç çeşitli sıklıklarda kullanılır, bunlar aşağıda kısaca açıklanmıştır. Terim belirteçlerinin tam listesi için GCC ve GAS kılavuzlarına başvurabilirsiniz.

Sık kullanılan belirteçler:

Yazmaç terimi belirteçleri (r)

Terimler, bu belirteç kullanılarak belirtilirse Genel Amaçlı Yazmaçlarda (GPR) tutulur. Aşağıdaki örneğe bakın:

```
asm ("movl %%cr3, %0\n" :"=r"(cr3val));
```

Burada `cr3val` değişkeni yazmaçta tutulur, `%cr3`'ün değeri bu yazmaca kopyalanır ve `cr3val`'ın değeri bellekte bu yazmaç tarafından güncelleştirilir. `r` belirteci belirtildiğinde GCC `cr3val`'ın değerini uygun herhangi bir Genel Amaçlı Yazmaçta (GPR) tutar. Yazmacı belirtmek için, yazmaç isimlerini doğrudan özel yazmaç belirteçleri ile belirtmelisiniz.

```
  a     %eax
  b     %ebx
  c     %ecx
  d     %edx
  S     %esi
  D     %edi
```

Bellek terimi belirteçleri (m)

Terimler bellekteyken, onlar üzerinde yapılacak her hangi bir işlem doğrudan bellek bölümüne de uygulanır, yazmaç belirteçleri ise değeri önce bir yazmaçta tutarlar sonra bellek bölümüne yazarlar. Yazmaç belirteçleri genelde kesin olarak gerektiklerinde veya önemli bir performans artışı sağladıklarında kullanılırlar. C değişkenlerinin "asm" içinde güncellenmesi gerektiğinde veya değerini bir yazmaçta tutmak istemediğiniz durumlarda bellek belirteçleri çok daha verimli olurlar. Örneğin `sdtr`'nin değeri `loc` bellek bölümünde tutuluyor:

```
("sidt %0\n" : :"m"(loc));
```

Karşılaştırma (rakam) belirteçleri

Bazı durumlarda tek bir değişken hem girdi hem de çıktı terimi olabilir. Bu durum "asm"de karşılaştırma belirteçleri ile belirtilir.

```
asm ("incl %0" :"=a"(var):"0"(var));
```

Örneğimizde belirteçleri karşılaştırmak için `%eax` yazmacı hem girdi hem de çıktı yazmacı olarak kullanılmıştır. `var` girdisi `%eax` yazmacına atanıyor, ve `%eax` arttırdıktan sonra `var`'da saklanıyor. " 0 " burada 0'ıncı çıktı belirteci gibi tanımlanmıştır. Bu, `var`'ın değerinin sadece `%eax`'te saklanabileceğini belirtir. Bu belirteç:

- Girdinin bir değişkenden okunduğu ve sonucun yine bu değişkene yazıldığı gibi durumlarda
- Girdi ve çıktıların ayrılmasının gerekmediği gibi durumlarda kullanılabilir. Karşılaştırma belirteçlerinin önemi geçerli yazmaçları verimli kullanmamızı sağlamalarındandır.

8. Genel Satıriçi Sembolik Makina Dili Kullanım Örnekleri

Aşağıdaki örnekler değişik terim belirteçlerinin kullanımını göstermektedir. Örnek verilebilecek çok fazla belirteç vardır fakat burada daha sık kullanılan belirteç türlerine yer verilmiştir.

"asm" ve register belirteci "r"

İlk önce 'r' belirtecinin "asm" de kullanılışına bir bakalım. Örneğimiz GCC'nin nasıl yazmaç ayırdığını ve değerleri nasıl güncelleştirdiğini göstermektedir.

```
int main(void)
{
        int x = 10, y;

        asm ("movl %1, %%eax;
             "movl %%eax, %0;"
                :"=r"(y)        /* y çıktı terimidir */
                :"r"(x)         /* x girdi terimidir */
                :"%eax");       /* %eax geri dönen yazmaç */
}
```

Bu örnekte x'in değeri "asm" içinde y'ye kopyalandı. x ve y "asm"den yazmaçların içinde geçtiler. Bunun için üretilecek Sembolik Makina Dili kodu şuna benzer:

```
    main:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        movl $10,-4(%ebp)
        movl -4(%ebp),%edx      /* x=10 %edx te tutulur */
#APP    /* asm burada başlıyor */
        movl %edx, %eax         /* x %eax e taşınır*/
        movl %eax, %edx         /* y edx e taşınır ve güncelleştirilir */

#NO_APP /* asm burada bitiyor */
        movl %edx,-8(%ebp)      /* y nin yığıttaki değeri %edx in değeri ile
                                   güncelleştirilir.*/
```

'r' belirteci kullanıldığında GCC herhangi bir yazmacı ayırmakta serbesttir. Örneğimizde `x`'i tutmak için `%edx`'i seçmiştir. `x`'in değerini `%edx`'ten okuduktan sonra `y` için yine `%edx`'i seçmiştir.

`y`'nin değeri çıktı terimi bölümünde oldukça `%edx`'in güncelleştirilen değeri `;y`'nin yığıttaki yeri, `-8(%ebp)`'de belirtilir. Eğer `y` girdi bölümünde tanımlanmış olsa idi, geçici `y` yazmacı (%edx) güncelleştirilmesine rağmen `y`'nin yığıttaki yeri güncelleştirilmezdi.

`%eax` geri dönen yazmaçlar listesinde oldukça GCC onu bilgi tutmak dışında kullanmayacaktır.

Çıktılar oluşmadan önce girdilerin yok olduğu farz edilerek, hem `x` girdisi hem `y` çıktısı aynı `%edx` yazmacında tutuldular. Ama şunu unutmayın eğer bir kaç komut işletecekseniz bunu yapamazsınız. Girdi ve çıktıların farklı yazmaçlarda tutulduğundan emin olmak için, & belirteç değiştiricisini kullanabiliriz.

Bununla ilgili bir örnek:

```
        int main(void)
{
        int x = 10, y;

        asm ("movl %1, %%eax;
             "movl %%eax, %0;"
                :"=&r"(y)       /* y çıktı terimidir ve
                                   & belirteç değiştiricisidir. */
                :"r"(x)         /* x girdi terimidir */
                :"%eax");       /* %eax geri dönen yazmaçtır*/
}
```

Ve burada bu örnek için üretilmiş Sembolik Makina Dili kodunu bulabilirsiniz, `x` ve `y`'nin "asm" de farklı yazmaçta tutulduğu görülmektedir.

```
    main:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        movl $10,-4(%ebp)
        movl -4(%ebp),%ecx      /* x, girdi %ecx tedir */
#APP
        movl %ecx, %eax
        movl %eax, %edx         /* y, çıktı %edx tedir */

#NO_APP
        movl %edx,-8(%ebp)
```

Özel yazmaç belirteçlerinin kullanımı

Şimdi kişisel yazmaçları terimler için belirteç olarak kullanmaya bir bakalım. Aşağıdaki örneğimizde `cpuid` komutu girdiyi `%eax` yazmacından alıyor ve çıktıyı dört yazmaca veriyor: `%eax`, `%ebx`, `%ecx`, `%edx`. `cpuid` girdisi, `op` değişkeni, "asm"ye `%eax cpuid`'nin de beklediği gibi yazmacından giriyor. `a`, `b`, `c` ve `d` belirteçleri çıktıda dört yazmaçtaki değerleri kendilerinde toplamak için kullanılmıştır.

```
    asm ("cpuid"
                : "=a" (_eax),
                  "=b" (_ebx),
                  "=c" (_ecx),
                  "=d" (_edx)
                : "a" (op));
```

Aşağıda bunun için üretilmiş Sembolik Makina Dili kodunu görebilirsiniz (_eax ,_ebx vb ... değişkenlerin yığıtta bulunduğu varsayılmıştır):

```
        movl -20(%ebp),%eax     /* 'op' yi %eax te tut -- girdi */
#APP
        cpuid
#NO_APP
        movl %eax,-4(%ebp)      /* %eax i _eax te tutar -- çıktı */
        movl %ebx,-8(%ebp)      /* diğer yazmaçları kendi çıktı
        movl %ecx,-12(%ebp)        değişkenlerinde tutar */
        movl %edx,-16(%ebp)
```

Aşağıdaki yolda `strcpy` fonksiyonu `"S"` ve `"D"` belirteçleri kullanılarak uygulanabilir:

```
    asm ("cld\n
              rep\n
              movsb"
              : /* girdi yok */
              :"S"(src), "D"(dst), "c"(count));
```

Kaynak gösterge `src %esi`'ye `"S"` belirteci ve hedef gösterge `dst`'ye `"D"` belirteci kullanılarak yerleştirilmiştir. Sayaç değeri `rep` önekinin gerektirdiği gibi `%ecx`'e yerleştirilmiştir.

Ve burada da 32–bit kodları birleştirip 64–bit kod elde etmek için, `%eax` ve `%edx` olmak üzere iki yazmaç kullanan bir belirteç göreceksiniz:

```
#define rdtscll(val) \
     __asm__ __volatile__ ("rdtsc" : "=A" (val))
```

Üretilen Sembolik Makina Dili kodu şuna benzer (`val` 64 bit bellek alanına sahipse):

```
#APP
        rdtsc
#NO_APP
        movl %eax,-8(%ebp)      /* A belirtecinin sonucu olarak
        movl %edx,-4(%ebp)         %eax ve %edx çıktı gibi çalışır */
```

`%edx:%eax`'in içindeki değerler 64 bit çıktı gibi çalışır.

Karşılaştırma belirteçlerinin kullanımı

Burada da dört parametreli sistem çağrıları için kodları göreceğiz:

```
#define _syscall4(type,name,type1,arg1,type2,arg2,type3,arg3,type4,arg4) \
type name (type1 arg1, type2 arg2, type3 arg3, type4 arg4) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
        : "=a" (__res) \
```

```
        : "0" (__NR_##name),"b" ((long)(arg1)),"c" ((long)(arg2)), \
          "d" ((long)(arg3)),"S" ((long)(arg4))); \
__syscall_return(type,__res); \
}
```

Üstteki örnekte sistem çağrısına dört argüman `%ebx, %ecx, %edx` ve `%esi`'ye `b, c, d` ve `S` belirteçleri kullanılarak bırakılmıştır. `"=a"` belirteci çıktıda kullanılmıştır. Bundan dolayı sistem çağrısının dönen değeri `%eax`, `__res` değişkenine aktarılır. Karşılaştırma belirteçlerinden `"0"`'ı girdi bölümünün ilk teriminin belirteci olarak kullanarak, sistem çağrısı numarası `__NR_##name %eax`'e atanır ve sistem çağrısın girdisi olarak işlem görür. Böylece `%eax` burada hem girdi hem de çıktı yazmacı olarak çalışır. Bunun için iki farklı yazmaç kullanılmadı. Ama şu unutulmamalıdır ki girdi (sistem çağrı numarası) çıkış üretilmeden önce kullanılmalıdır.

Bellek terimi belirtecinin kullanımı

Aşağıdaki küçük azaltma işlemini inceleyin:

```
    __asm__ __volatile__(
                "lock; decl %0"
                :"=m" (counter)
                :"m" (counter));
```

Bunun için üretilen Sembolik Makina Dili kodu şöyle olur:

```
    #APP
        lock
        decl -24(%ebp) /* sayaç bu bellek bölümünde değiştirilir. */
#NO_APP.
```

Burada sayaç için yazmaç belirteci kullanmadan önce bir düşünmelisiniz. Eğer kullanırsanız, önce sayacın içeriğini bir yazmaca atamalısınız. Azaltma işlemini yaptıktan sonra sonucu belleğine atamalısınız. Ama o zaman da `lock` kullanma ve kodu küçük tutma çabamızı boşa çıkarmış oluruz. Bu bellek belirtecinin gereğini kesin olarak göstermektedir.

Geri Dönen yazmaçların Kullanımı

Temel Bellek Kopyasının işlemesini inceleyin.

```
        asm ("movl $count, %%ecx;
              up: lodsl;
              stosl;
              loop up;"
                :                       /* çıktı yok */
                :"S"(src), "D"(dst)     /* girdi */
                :"%ecx", "%eax" );      /* geri dönen yazmaçlar */
```

`lodsl %eax`'i değiştirirken, `lodsl` ve `stosl` komutları onu dolaylı olarak kullanır. `%ecx` yazmacı doğrudan `count`'u çağırır. Biz `%eax` ve `%ecx`'i geri dönen yazmaçlar listesinde belirtmediğimiz sürece GCC bunun farkında olmayacaktır. Bunu yapmadığımız sürecce GCC `%eax` ve `%ecx`'in serbest olduğunu varsayar ve onları başka bilgiler tutmak için kullanabilir. Şunu bilmelisiniz ki `%esi` ve `%edi` "asm" tarafından kullanılıyor ve geri dönen yazmaçlar arasında belirtilmemişlerdir. Çünkü onlar zaten girdi terimlerinin listesinde bulunmaktadırlar. Ve son olarak, eğer dolaylı veya doğrudan kullanılmış bir yazmaç girdi veya çıktı listesinde bulunmuyorsa, onu geri dönen yazmaçlar listesine yazmalısınız.

9. Sonuç

Şu sonuca varabiliriz ki Satıriçi Sembolik Makina Dili çok geniştir ve burada değinemediğimiz bir çok olanak da sağlar. Ama bu yazıdan temeli kavramanız halinde kendi başınıza Satıriçi Sembolik Makina Dili ile kod yazmaya başlayabilirsiniz.

10. Yararlanılan Kaynaklar

Using and Porting the GNU Compiler Collection (GCC) manual[B3].

GNU Assembler (GAS) manual[B4].

Brennan's Guide to Inline Assembly[B5].

11. Yazar Hakkında

Bharata B. Rao Hindistan Mysore Üniversitesi Elektronik ve Haberleşme Mühendisliği bölümünü okudu. Hindistan'da 1999'dan beri IBM Global Services için çalışıyor. IBM Linux Teknoloji Merkezi üyesidir. İlk olarak Linux RAS (Reliability, Availability, and Serviceability) Projesinde bulunmuştur. Diğer hobileri İşletim Sistemleri ve İşlemci mimarisidir. Ona <rbharata (at) in.ibm.com>'dan ulaşabilirsiniz.

Notlar

a) Belge içinde dipnotlar ve dış bağlantılar varsa, bunlarla ilgili bilgiler bulundukları sayfanın sonunda dipnot olarak verilmeyip, hepsi toplu olarak burada listelenmiş olacaktır.

b) Konsol görüntüsünü temsil eden sarı zeminli alanlarda metin genişliğine sığmayan satırların sığmayan kısmı ¬ karakteri kullanılarak bir alt satıra indirilmiştir. Sarı zeminli alanlarda ¬ karakteri ile başlayan satırlar bir önceki satırın devamı olarak ele alınmalıdır.

(B1) `../howto/gpl.pdf`

(B3) `http://gcc.gnu.org/onlinedocs/gcc_toc.html`

(B4) `http://www.gnu.org/manual/gas-«2.9.1/as.html`

(B5) `http://www.delorie.com/djgpp/doc/brennan/brennan_att_inline_djgpp.html`

Bu dosya (linux–inline–assembly.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

6 Şubat 2007