TESTO della LEZIONE di Venerdì 29 novembre 2013 su
GESTIONE DELLA MEMORIA DA PARTE DEL SISTEMA OPERATIVO

In questa parte di lezione vedremo come vengono trattati gli indirizzi durante la produzione e poi durante l'esecuzione di un file eseguibile. Analizzeremo in particolare come vengono trattati ed espressi gli indirizzi nelle seguenti situazioni:
1) Nei singoli moduli oggetto compilati, in cui c'è visibilità di un solo modulo alla volta, e in cui si considerano gli indirizzi LOGICI, ovvero indirizzi che specificano posizioni nel modulo su file e rispetto all'inizio del rispettivo segmento.
2) Nell'eseguibile creato dal linker, in cui sono stati accorpati i diversi moduli oggetto ed accorpati le corrispondenti sezioni dei diversi moduli, e in cui si considerano gli indirizzi LOGICI, ovvero indirizzi che specificano posizioni nell'eseguibile su file e rispetto all'inizio del rispettivo segmento. Quando il linker crea l'eseguibile a partire dai moduli oggetto, il linker effettua la RILOCAZIONE STATICA degli indirizzi assoluti.
3) Durante l'esecuzione dell'eseguibile, in particolare non appena il loader ha terminato il caricamento in memoria (virtuale o fisica) dell'eseguibile. Durante il caricamento, il loader effettua la RILOCAZIONE DINAMICA degli indirizzi, in particolare decide dove collocare in memoria i segmenti dell'eseguibile, completato le tabelle (del sistema operativo) dei segmenti inserendo le descrizioni e le posizioni dei segmenti dell'eseguibile, assegna ai registri di segmento il valore degli indirizzi di inizio dei segmenti a cui si riferiscono. Se il sistema operativo NON USA la paginazione, il loader carica effettivamente in memoria fisica i segmenti e quindi gli indirizzi a cui ci si riferisce sono INDIRIZZI FISICI, cioè indirizzi di celle RAM fisica. Se invece il sistema operativo USA LA PAGINAZIONE, il caricamento viene effettuato non in memoria fisica bensì in memoria virtuale, cioè l'eseguibile viene caricato nella memoria swappata su disco ed allora gli indirizzi a cui ci si riferisce non sono indirizzi fisici bensì INDIRIZZI VIRTUALI (detti anche lineari).
4) Durante l'esecuzione dell'eseguibile, per quei sistemi operativi che prevedono la paginazione, le pagine che servono all'esecuzione vengono caricate da disco e collocate in memoria ed allora gli indirizzi a cui ci si riferisce sono indirizzi FISICI.

INDIRIZZI
In un processore come quelli della famiglia Intel, **l'indirizzo di una locazione di memoria è specificato mediante due contributi, il primo che specifica l'inizio del segmento in cui si trova la locazione di memoria e il secondo che specifica lo scostamento (offset) della locazione di memoria rispetto all'inizio del segmento.**
Per quanto riguarda la sezione text (il codice), l'inizio del segmento viene specificato dal registro CS (Code Segment register) mentre lo scostamento viene indicato dal registro IP (Instruction Pointer). La somma (vedi più avanti l'argomento SEGMENTAZIONE per ulteriori dettagli) dei due contributi (solitamente denotata col termine Program Counter) indica la locazione di memoria in cui si trova la prossima istruzione da eseguire.
Per la sezione data, il registro di segmento usato è DS (Data Segment). Infine, per la sezione stack, il registro di segmento usato è SS (Stack Segment) mentre il registro SP (Stack Pointer) contiene lo scostamento della cima dello stack rispetto all'inizio del segmento di stack.

ISTRUZIONI DI SALTO.
In un eseguibile, formato da istruzioni macchina, dopo l'esecuzione di una istruzione si passa solitamente all'esecuzione della istruzione successiva, cioè quella che segue, nel file, l'istruzione appena eseguita. Questo è vero tranne nel caso delle istruzioni dette "di salto" (JUMP) le quali invece specificano esplicitamente quale è la prossima istruzione da eseguire. Le istruzioni di salto vengono impiegate per implementare i costrutti linguistici quali le iterazioni (for, while) e i costrutti condizionali (if, else, switch) ed anche le chiamate a funzione.

EFFETTO DI UNA ISTRUZIONE DI SALTO.
L'effetto di una istruzione di salto è di modificare il contenuto del registro IP (Instruction Pointer) ed in qualche caso anche del registro CS (Code Segment) in modo da indicare la prossima istruzione macchina da eseguire.

ISTRUZIONI DI SALTO CONDIZIONALE E NON CONDIZIONALE
Una istruzione di salto può essere dipendente da una specifica condizione ed in tal caso viene detto "Salto Condizionale". Se la condizione è vera l'istruzione fa saltare all'indirizzo specificato dall'istruzione stessa, se invece la condizione è falsa il salto non viene effettuato e l'istruzione che verrà eseguita dopo l'istruzione di salto sarà l'istruzione che nell'eseguibile si trova dopo l'istruzione di salto stesso.
Un salto non condizionale, invece, specifica la prossima istruzione senza richiedere alcuna condizione.
La condizione da verificare di solito è il valore di alcuni bit di un registro detto FLAGS. Questi bit sono settati durante l'esecuzione di alcune istruzioni, ad esempio le istruzioni di confronto. Ad esempio, l'istruzione CMP AX, 0 controlla se il registro AX contiene il valore 0 oppure no, e a seconda del risultato del confronto setta a 1 o a 0 il valore di alcuni bit del registro FLAGS. Una istruzione di salto condizionale controlla il valore dei bit del registro FLAGS e verifica la condizione. Ad esempio l'istruzione JZ (Jump if Zero) fa saltare all'indirizzo specificato se i bit del registro EFLAGS indicano che il confronto precedente ha verificato la presenza di uno zero.
Quindi la seguente coppia di istruzioni

```
CMP AX, 0
JZ      label_destinazione
```

fa saltare all'indirizzo *label_destinazione* se dentro il registro AX c'è uno zero. Infatti, l'istruzione cmp effettua il confronto e setta i bit del registro FLAGS a seconda del risultato del confronto. Poi l'istruzione JZ legge i bit del registro FLAGS ed effettua il salto a seconda del valore di questi bit.

LA MEMORIA DAL PUNTO DI VISTA DEI MODULI OGGETTO.
Un modulo oggetto è un file che contiene codice macchina ed è stato ottenuto o dalla compilazione di un codice sorgente in linguaggio C oppure è stato ottenuto dall'assemblaggio di un codice sorgente in linguaggio assembly. Il modulo oggetto è solo una parte dell'eseguibile finale, quindi il modulo oggetto vede solo il proprio contenuto e non quello degli altri moduli che comporranno l'eseguibile finale. Quindi, la sezione text di ciascun modulo oggetto crede di occupare la parte di memoria che comincia all'indirizzo ZERO. In altre parole, ciascun modulo crede che il proprio segmento di codice inizi all'indirizzo ZERO e che quindi il Code Segment valga ZERO. In questo modo, l'indirizzo di ciascuna istruzione macchina all'interno del modulo è sia l'indirizzo assoluto e sia lo scostamento rispetto all'inizio del segmento.

IPOTESI IMPORTANTE – PER RENDERE PIU' SEMPLICE LA TRATTAZIONE DEGLI ARGOMENTI, D'ORA IN AVANTI GUARDIAMO IL CODICE MACCHINA DI UN MODULO TRADUCENDOLO NELL'EQUIVALENTE CODICE ASSEMBLY PER RENDERE ILCODICE LEGGIBILE. QUINDI NEGLI ESEMPI FORNITI, INVECE DI SCRIVERE CODICE MACCHINA SCRIVEREMO CODICE ASSEMBLY CHE SE FOSSE ASSEMBLATO GENEREREBBE ILCODICE MACCHINA DI CUI DESCRIVIAMO GLI EFFETTI.

LE ETICHETTE ALL'INTERNO DI UN MODULO OGGETTO
Le etichette (o label) sono degli identificatori univoci seguiti da un carattere due punti (:), collocati all'inizio delle istruzioni assembly, che servono a riferirsi (identificare) l'indirizzo in cui l'istruzione si trova. Ad esempio la seguente istruzione

```
ZZ:   INC BX
```

identifica l'indirizzo in cui si trova l'istruzione "INC BX" mediante l'etichetta ZZ.
Un'etichetta di un'istruzione eseguibile serve ad effettuare un salto verso quella istruzione. Un'etichetta di una variabile nella sezione "data" invece serve ad indicare l'indirizzo della variabile e ad effettuare degli assegnamenti da/su quella variabile.
Un'etichetta può essere utilizzata sia dall'interno del modulo in cui l'etichetta è contenuta ma anche da altri moduli esterni.

ESEMPIO
Supponiamo di avere due moduli oggetto, chiamati modulo A e modulo B e consideriamo delle porzioni del codice assembly che ha generato questi moduli. Sull'estrema sinistra abbiamo aggiunto gli indirizzi assoluti in cui si trovano le istruzioni, seguono le etichette con in fondo il carattere due punti (:) ed infine a destra l'istruzione vera e propria. Nel codice che segue, i tre puntini in sequenza … indicano che lì c'è altro codice, che non interessa la trattazione e che quindi non viene inserito.

MODULO A

```
   0          . . .
 100          PUSH  BP
 150          CMP  AX,  8
 200          JNE   ZZ
 250          . . .
. . .         . . .
1000          SUB AX, 9
1050          ADD SP, 12
              . . .
1300          MOV  BP, SP
. . .         . . .
```

Fine MODULO A

MODULO B

```
   0      . . .
 100          ADD BP. 8
 150          PUSH  12
 200          JMP   1000
 . . .        . . .
 950    ZZ:   INC BX
1000          MOV AX, BX
1050          JRE  250
. . .         . . .
1300          MOV BP, AX
. . .         . . .
```

Fine MODULO B

ISTRUZIONI DI SALTO RELATIVO
Un salto relativo è un salto che definisce la posizione della prossima istruzione da eseguire specificando lo scostamento (positivo o negativo) di quella istruzione rispetto alla posizione della attuale istruzione in esecuzione (che è l'istruzione di salto stessa). Nell'esempio del modulo B, all'indirizzo 1050 troviamo l'istruzione di salto relativo JRE 250 che specifica l'operando 250. In tal modo l'istruzione di salto relativo specifica che la prossima istruzione da eseguire si trova 250 byte più avanti rispetto alla istruzione attualmente in esecuzione. Poiché l'attuale istruzione si trova all'indirizzo 1050, la prossima istruzione sarà quella all'indirizzo 1050+250=1300, e quindi sarà l'istruzione MOV BP, AX.

La domanda da porsi è la seguente: durante l'esecuzione del programma, come fa l'istruzione di salto a sapere dove si trova lei stessa? La risposta è semplice: nel momento in cui eseguo una istruzione, il registro IP contiene l'indirizzo della istruzione in esecuzione. Quindi, il compito dell'istruzione di salto relativo è solo quello di modificare il contenuto del registro IP aggiungendovi l'offset specificato dal salto relativo. Nell'esempio del modulo B, all'indirizzo 1050 troviamo l'istruzione di salto relativo JRE 250 che specifica l'operando 250. Affinché venga eseguita questa istruzione, è necessario che nel registro IP ci sia il valore 1050. L'istruzione JRE 250 aggiunge il valore 250 al contenuto del registro IP facendolo diventare 1300. Quindi la successiva istruzione che sarà eseguita sarà quella all'indirizzo 1300, cioè l'istruzione MOV BP, AX.

ISTRUZIONI DI SALTO ASSOLUTO

Un salto assoluto è un salto che definisce la posizione della prossima istruzione da eseguire specificando l'indirizzo assoluto di quella prossima istruzione e non lo scostamento di quella istruzione rispetto all'istruzione attuale. Nell'esempio del modulo B, all'indirizzo 200 troviamo l'istruzione di salto assoluto JMP 1000 che specifica l'operando 1000. In tal modo l'istruzione di salto assoluto specifica che la prossima istruzione da eseguire si trova all'indirizzo 1000, indipendentemente dalla istruzione attualmente in esecuzione. Quindi la prossima istruzione sarà quella all'indirizzo 1000, e quindi sarà l'istruzione MOV AX, BX. L'istruzione di salto assoluto modifica il contenuto del registro IP, copiando nel registro il valore dell'operando (l'indirizzo) specificato dall'istruzione di salto, nell'esempio in IP viene copiato 1000.
I salti assoluti possono specificare un indirizzo in due diversi modi:
i) **numericamente**, mediante un valore numerico, come nell'esempio JMP 1000 in cui si specifica l'indirizzo (100) della prossima istruzione,
ii) **simbolicamente**, mediante un'etichetta, come nell'esempio del modulo A all'indirizzo 200 dove si trova l'istruzione JMP ZZ. In questo secondo esempio l'indirizzo dell'istruzione viene specificato mediante l'etichetta (l'identificatore ZZ) dell'istruzione a cui saltare.

Va messo in evidenza che un salto assoluto può specificare anche un indirizzo di un altro modulo, come nel caso dell'istruzione JNE ZZ contenuta nel modulo A che ordina un salto ad un indirizzo del modulo B. In casi come questo, l'indirizzo ZZ non può essere risolto nella fase di compilazione, poiché il compilatore vede solo un modulo alla volta. Il compilatore, quando compila il modulo A, non potendo sapere dove si trova l'etichetta ZZ, mette questo simbolo ZZ nella tabella dei simboli non risolti che include nel modulo oggetto generato dal modulo sorgente A. In tal modo, quando il linker processerà i moduli oggetto A.o e B.o leggerà quali sono i simboli non risolti e cercherà di trovare a quali indirizzi si riferiscono, scoprendo così che ZZ si riferisce ad una locazione del modulo B e inserendo nel file eseguibile l'indirizzo della istruzione etichettata con ZZ al posto del simbolo ZZ nella istruzione di salto JNE ZZ.

SALTI NEAR E FAR (VICINI e LONTANI)

Le istruzioni assembly includono istruzioni che effettuano salti cosiddetti NEAR (vicini) e FAR (lontani). I salti NEAR permettono di saltare a posizioni vicine a quelle in cui ci si trova, poiché permettono di specificare degli offset piccoli, in quanto si possono usare come operando che specifica il salto solo pochi bytes o addirittura uno solo. Invece i salti FAR usano come operando che specifica il salto un intero grande (cioè composto da molti bit) e quindi permettono di specificare un offset grande. Addirittura alcuni salti FAR permettono di specificare oltre ad un nuovo offset (da mettere nello Instruction Pointer) anche un nuovo indirizzo di inizio segmento (da mettere nel Code Segment) per poter addirittura cambiare il segmento di codice da usare.

FASE DI LINKING E RILOCAZIONE STATICA DEGLI INDIRIZZI nel codice e nei dati.

Quando il linker collega più moduli oggetto per formare l'eseguibile finale, deve mettere assieme dei moduli ognuno dei quali è stato costruito come se le proprie istruzioni iniziassero tutte dall'indirizzo ZERO. Accorpando i moduli, uno di questi verrà collocato all'inizio dell'eseguibile come se le sue istruzioni cominciassero all'indirizzo ZERO, ma gli altri moduli verranno posizionati dopo la fine del primo e/o dopo la fine degli altri moduli, trovandosi così in una posizione in cui la prima istruzione del modulo non comincia all'indirizzo ZERO. La RILOCAZIONE STATICA è l'operazione svolta dal linker che modifica le istruzioni di salto cambiando l'operando che specifica l'indirizzo a cui saltare per specificare correttamente l'indirizzo in cui si trova l'istruzione da eseguire ora che i diversi moduli sono stati accorpati. Questa rilocazione viene detta STATICA poiché viene eseguita una volta sola, dal linker, cioè quando si genera l'eseguibile partendo dai moduli oggetto. La rilocazione statica riguarda tutti i salti assoluti, sia quei salti che specificano indirizzi numerici sia quelli che specificano indirizzi simbolici (label). Le considerazioni valide per i salti con indirizzi simbolici valgono anche per le istruzioni che specificano delle variabili globali (cioè nella sezione data) mediante delle etichette.

ESEMPIO DI FILE ESEGUIBILE OTTENUTO DAL LINKING DEI MODULI A e B
Supponiamo di avere i due moduli oggetto A e B, descritti prima. Supponiamo che il linker abbia accorpato i due moduli A e B per generare l'eseguibile mettendo prima A e poi B. Supponiamo che il modulo A sia grande 2000 bytes e che quindi il modulo B sia stato messo dopo la fine del modulo A e cioè che cominci all'indirizzo 2000. Il file eseguibile che otteniamo dopo la fase di linking è descritto qui di seguito (come prima usiamo codice assembly invece di codice macchina, per facilitare la comprensione)

FILE ESEGUIBILE

```
;   parte originate dal modulo A
    0          . . .
  100          PUSH  BP
  150          CMP  AX,  8
  200          JNE    2950
  250          . . .
 . . .         . . .
1000           SUB AX, 9
1050           ADD SP, 12
               . . .
1300           MOV  BP, SP
. . .          . . .
;   parte originate dal modulo B, inizia all'indirizzo 2000
2000           . . .
2100           ADD BP. 8
2150           PUSH  12
2200           JMP    3000
  . . .         . . .
2950    ZZ:    INC BX
3000           MOV AX, BX
3050           JRE  250
. . .          . . .
3300           MOV BP, AX
. . .          . . .
Fine Eseguibile
```

LA RILOCAZIONE STATICA NON MODIFICA LE ISTRUZIONI DI SALTO RELATIVO.
Consideriamo un esempio di salto relativo, quello che troviamo all'indirizzo 1050 nel modulo B e che ci fa saltare 250 byte più avanti (JRE 250) fino all'indirizzo 1300. Supponiamo che il linker abbia accorpato i due moduli A e B per generare l'eseguibile mettendo prima A e poi B. Supponiamo che il modulo A sia grande 2000 bytes e che quindi il modulo B sia stato messo dopo la fine del modulo A e cioè che cominci all'indirizzo 2000. L'istruzione di salto che si trovava all'indirizzo 1050 del modulo B e che ci faceva saltare all'indirizzo 1300 ora si trova all'indirizzo 3050 e deve farci saltare all'indirizzo 3300. Il linker non deve fare nulla perché l'istruzione di salto relativo, eseguita quando IP vale 3050, aggiunge 250 all'IP e quindi fa saltare all'indirizzo IP=3050+250=3300, cioè all'indirizzo giusto. Da ciò si capisce che le istruzioni di salto relativo non necessitano di essere modificate durante il linking.

LA RILOCAZIONE STATICA MODIFICA LE ISTRUZIONI DI SALTO ASSOLUTO CON OPERANDO NUMERICO.
Consideriamo un esempio di salto assoluto con offset numerico, quello che troviamo all'indirizzo 200 nel modulo B e che ci fa saltare all'indirizzo 1000 dello stesso modulo B. Supponiamo come prima che il linker abbia accorpato i due moduli A e B per generare l'eseguibile mettendo prima A e poi B. Supponiamo come prima che il modulo A sia grande 2000 bytes e che quindi il modulo B sia stato messo dopo la fine del modulo A e cioè che cominci all'indirizzo 2000. L'istruzione di salto che si trovava all'indirizzo 200 del modulo B e che ci faceva saltare all'indirizzo 1000 ora si trova all'indirizzo 2200 e deve farci saltare all'indirizzo 3000. Il linker perciò deve modificare l'operando del salto assoluto aggiungendo al vecchio indirizzo assoluto (1000) il valore dell'indirizzo di inizio del modulo B nell'eseguibile (2000). Il linker quindi sostituisce 3000 a 1000 nella istruzione di salto, che diventa JMP 3000 e che ci fa saltare all'istruzione corretta (**MOV BP, AX**). Da ciò si capisce che le istruzioni di salto assoluto devono essere modificate durante il linking operando la rilocazione statica.

LA RILOCAZIONE STATICA MODIFICA LE ISTRUZIONI DI SALTO ASSOLUTO CON OPERANDO SIMBOLICO (una label non risolta).
Consideriamo un esempio di salto assoluto con offset simbolico non risolto, cioè con un indirizzo che si trova in un altro modulo. L'esempio è quello che troviamo all'indirizzo 200 nel modulo A e che ci fa saltare all'istruzione identificata dalla etichetta ZZ dell'altro modulo B. Supponiamo come prima che il linker abbia accorpato i due moduli A e B per generare l'eseguibile mettendo prima A e poi B. Supponiamo come prima che il modulo A sia grande 2000 bytes e che quindi il modulo B sia stato messo dopo la fine del modulo A e cioè che cominci all'indirizzo 2000. L'istruzione di salto (JNE ZZ) si trovava all'indirizzo 200 del modulo A e che ci faceva saltare all'indirizzo 950 del modulo B dove c'era l'etichetta ZZ. Dopo il linking l'istruzione di salto si trova ancora all'indirizzo 200 ma ora deve farci saltare all'indirizzo 2950. Il linker perciò deve modificare l'operando del salto assoluto guardando dove si trova ora l'istruzione con la label ZZ (si trova all'indirizzo 950+2000) aggiungendo al vecchio indirizzo assoluto della label ZZ (950) il valore dell'indirizzo di inizio del modulo B nell'eseguibile (2000). Il linker quindi sostituisce 2950 all'etichetta ZZ nella istruzione di salto, che diventa JNE 2950 e che ci fa saltare all'istruzione corretta (**MOV AX, BX**). Da ciò si capisce che le istruzioni di salto assoluto che specificano un indirizzo simbolico (label) di un altro modulo, devono essere modificate durante il linking effettuando la rilocazione statica degli indirizzi.

FINE DELLA FASE DI LINKING.
Al termine della fase di linking, se questa non ha prodotto degli errori, il linker salva su disco un file contenente l'eseguibile ottenuto dal collegamento dei diversi moduli.

LOADING DEL PROGRAMMA IN MEMORIA.

Quando l'utente, da una shell di comandi, ordina l'esecuzione di un programma eseguibile, il sistema operativo utilizza una sua funzionalità, detta LOADER, per:
- caricare il programma in memoria RAM creando i vari segmenti (test,data, stack) del programma.
- scrivere in alcune tabelle del sistema operativo alcune informazioni sui segmenti creati (dove si trovano in memoria, quanto sono grandi, chi ha i permessi di scrivere e leggere dentro i segmenti).
- copiare nel segmento text il codice macchina del programma, copiare nella sezione data i dati del programma).
- Inserire nel registro di segmento SS l'indirizzo di inizio del segmento Stack e nel registro SP l'offset pari alla cima dello stack.
- Inserire nel registro di segmento DS l'indirizzo di inizio del segmento Data.
- Inserire nel registro di segmento CS l'indirizzo di inizio del segmento text e nel registro IP l'offset della prima istruzione da eseguire del programma.
- cominciare l'esecuzione della prima istruzione del programma indicata da CS:IP utilizzando lo stack indicato da SS:SP, ed usando la sezione data indicata da DS.

RILOCAZIONE DINAMICA DURANTE IL LOADING.
La rilocazione dinamica e' l'operazione, eseguita durante il caricamento del programma in memoria, mediante la quale si rende possibile che un programma possa essere eseguito in una posizione qualsiasi in memoria senza dover modificare le istruzioni macchina dei salti. Si ricordi che nei salti assoluti, l'indirizzo che viene specificato dalle istruzioni è l'offset rispetto all'inizio del segmento di codice. Ebbene, durante la rilocazione dinamica vengono specificati gli indirizzi di inizio in memoria dei diversi segmenti (codice, data e stack) e tali indirizzi vengono collocati nei registri di segmento CS, DS e SS. In tal modo un indirizzo che nel codice del programma era specificato mediante la coppia CS:offset si trova a puntare alla giusta istruzione perché si usa come contenuto del registro CS l'indirizzo di inizio del segmento di codice, dovunque questo sia stato collocato. Chiariamo il concetto guardando come esempio l'istruzione di salto JNE 2950 che nel programma su disco si trova all'indirizzo 200 e che dovrebbe far saltare all'istruzione INC BX che si trova all'indirizzo 2950 del programma su disco. Nel programma su disco questi salti assumono che il segmento di codice inizi all'indirizzo ZERO, cioè che CS valga ZERO. Durante il LOADING (il caricamento in memoria) del programma per l'esecuzione, il segmento di codice viene collocato da qualche parte e nel registro CS viene messo (vedi più avanti l'argomento SEGMENTAZIONE per ulteriori dettagli) l'indirizzo di inizio in RAM di quel segmento di codice. Supponiamo che il segmento di codice venga caricato a partire dall'indirizzo 4000. Allora in CS verrà messo l'indirizzo 4000 (vedi più avanti l'argomento SEGMENTAZIONE per ulteriori dettagli). L'istruzione di salto si troverà all'indirizzo assoluto 4000+200=4200 e l'istruzione a cui saltare si troverà all'indirizzo 4000+2950=6950. Dopo il LOADING, poiché ora CS contiene l'indirizzo di inizio del segmento di codice in memoria cioe 4000, l'istruzione di salto JNE 2950 salterà all'indirizzo specificato da CS e dall'offset 2950, cioè all'indirizzo 4000+2950=6950 che è esattamente l'indirizzo dell'istruzione INC BX a cui volevamo arrivare.
In definitiva, la rilocazione dinamica modifica solo i registri di segmento assegnando a questi, al caricamento, l'indirizzo di inizio dei segmenti a seconda di dove i segmenti sono stati collocati in memoria. In tal modo gli indirizzi specificati nel codice rimangono coerenti alla posizione in cui si trovano effettivamente le istruzioni a cui si vuole saltare.

SEGMENTAZIONE PURA E CALCOLO DEGLI INDIRIZZI FISICI IN MEMORIA NELL' 8088 e 8086
Prima sono stato volutamente impreciso affermando che "all'atto del LOADING in memoria dentro i registri di segmento vengono messi gli indirizzi di inizio dei segmenti". In realtà quello che viene messo nei registri di segmento è una funzione degli indirizzi di inizio dei segmenti, e in generale dipende dal processore che si usa.
In particolare, nei processori Intel 8088 e 8086 sono vere le seguenti tre affermazioni:

1) I segmenti possono iniziare solo ad indirizzi multipli di 16 (0, 16, 32, …..)
2) Dato un segmento (ad esempio di codice) che inizia all'indirizzo ADDRESS (che è multiplo di 16), nel registro di segmento (ad esempio CS) per specificare l'inizio di quel segmento, viene collocato il valore ADDRESS / 16.
3) Data una locazione di memoria (ad esempio un'istruzione macchina del codice) il cui indirizzo è specificata dalla coppia formata da un registro di segmento (ad esempio CS) e da un offset (OFFSET), allora l'8088 calcola l'indirizzo assoluto di quella locazione di memoria mediante la seguente formula: ADDRESS = CS*16+OFFSET
Cioè l'indirizzo di una locazione di memoria indicata dalla coppia REGISTRO_DI_SEGMENTO e OFFSET viene ottenuto moltiplicando per 16 il contenuto del REGISTRO_DI_SEGMENTO e poi sommando l'OFFSET. Questo calcolo viene effettuato rapidamente da una parte specializzata della CPU denominata MMU (Memory Management Unit).
Il MOTIVO per cui si opera in questo modo dipende dal fatto che il processore 8088 dispone di registri di segmento formati da 16 bit ed usa offset (contenuti in registri general-purpose) formati al massimo da 16 bit. Però l'8088 ha un bus degli indirizzi che permette di specificare degli indirizzi in memoria formati da 20bit e quindi potenzialmente di indirizzare una quantità di memoria pari a 2^20 bytes, cioè un MegaByte di memoria che è esattamente la memoria RAM di cui dispone il processore.
Cosi', moltiplicando il contenuto del registro di segmento per 16 ottengo un numero memorizzabile in 20 bit (la dimensione del bus degli indirizzi) e così riesco a piazzare l'inizio del segmento in un qualunque punto (purché multiplo di 16) di quel 1MB di memoria fisica. Poi aggiungendo l'offset posso accedere ai 2^16=65536 bytes dopo l'inizio del segmento. In questo modo posso accedere a tutto quel MB di memoria RAM.
INVECE, se il processore NON moltiplicasse per 16 il contenuto del registro di segmento, allora con i 16 bit del registro di segmento potrei specificare un indirizzo di inizio del segmento compreso tra 0 e 65535, e poi potrei spaziare di altri 65536 bytes aggiungendo l'offset a 16 bit. In totale potrei accedere a 2*65536=2*64KB invece che a 1024 KB.
In definitiva, moltiplicando per 16 il contenuto del registro di segmento e poi aggiungendo l'offset a 16 bit, riesco ad aumentare l'ampiezza della memoria RAM indirizzabile, al solo costo di perdere di granularità, cioè al solo costo di dover piazzare l'inizio di ciascun segmento ad un indirizzo multiplo di 16.

SEGMENTAZIONE PURA E CALCOLO DEGLI INDIRIZZI FISICI IN MEMORIA in IA-32
Nell'architettura IA-32 sono presenti registri di segmento formati da 16 bit e registri general-purpose formati da 32 bit. Il bus degli indirizzi è formato da 32 linee, quindi i processori della famiglia IA-32 possono indirizzare fino a 2^32 bytes, cioè circa 4GB.
L'indirizzamento può essere effettuato o mediante la segmentazione pura (cioè senza paginazione) oppure attivando la paginazione.
Nel caso più semplice, quello della segmentazione pura, il registro di segmento non contiene l'indirizzo di inizio del segmento diviso 16, bensì contiene un indice (detto selettore di segmento) che permette di accedere ad una posizione di un vettore di strutture mantenuto in memoria. Un apposito registro, Segment Table Base Register (STBR), contiene l'indirizzo fisico di inizio di tale vettore. Un altro registro, Segment Table Limit Register (STLR), contiene il numero di elementi del vettore. L'indice contenuto nel registro di segmento permette di accedere ad una posizione di questo vettore, leggendo cioè un elemento del vettore. L'elemento del vettore contiene le informazioni sul segmento a cui si vuole accedere. Queste informazioni includono l'indirizzo di inizio del segmento e la dimensione del segmento stesso.
Quando si cerca di ottenere l'indirizzo fisico di una locazione di memoria partendo dalla coppia (REGISTRO_DI_SEGMENTO_A_16_BIT : OFFSET_A_32_BIT) il processore svolge le seguenti tre operazioni:

1) Confronta l'indice contenuto nel registro di segmento per vedere se è minore del numero di elementi del vettore, in caso contrario genera un segmentation fault.
2) Accede all'elemento del vettore e controlla i permessi del segmento. Se i permessi non permettono l'accesso, il processore genera un segmentation fault.
3) Controlla la dimensione del segmento e la confronta con l'offset della locazione di memoria a cui si vuole accedere. Se l'offset supera il limite del segmento allora il processore genera un segmentation fault.
4) Legge nell'elemento del vettore il valore dell'indirizzo di inizio del segmento (in realtà l'indirizzo diviso 16, come nell'8088) e poi calcola l'indirizzo logico mediante la formula consueta:

BASE*16+OFFSET