ELABORATO ASSEMBLY

UniVr - Dipartimento di Informatica

2023/2024

Realizzato da:
Giuseppe Caltroni: VR489402
Samy El Ansari: VR501144

INDICE

1. INTRODUZIONE DEL PROGETTO

1.1. DESCRIZIONE DEL PROGETTO

- **CONSEGNA**
  Si sviluppi un software per la pianificazione delle attività di un sistema produttivo, per i successivi prodotti, fino ad un massimo di 10 prodotti, nelle successive 100 unità di tempo dette “slot temporali”. Il sistema produttivo produce un prodotto alla volta. La produzione è suddivisa in slot temporali uniformi, e durante ogni slot temporale solo un prodotto può essere in elaborazione.

- **STRUTTURA DI UN PRODOTTO**
  Ogni prodotto è caratterizzato da quattro valori interi:

Nome	Descrizione	Range
Identificativo	Il codice identificativo del prodotto	1 - 127
Durata	Indica il numero di slot temporali necessarie per completare il prodotto	1 - 10
Scadenza	Il tempo massimo in unita' di tempo entro cui il prodotto deve essere ultimato	1 - 100
Priorita'	Indica la proprita' con cui deve essere elaborato il prodotto	1 - 5

- **PENALTY**
  Per ogni prodotto completato in ritardo rispetto alla scadenza indicata, l’azienda dovrà pagare una penale in Euro pari al valore della priorità del prodotto completato in ritardo, moltiplicato per il numero di unità di tempo di ritardo rispetto alla sua scadenza.

  Ad esempio, se il prodotto:

  Identificativo: 4; Durata: 10; Scadenza: 25; Priorità: 4;

  venisse messo in produzione all’unità di tempo 21, il sistema completerebbe la sua produzione al tempo 30, con 5 unità di tempo di ritardo rispetto alla scadenza richiesta, l’azienda dovrebbe pagare una penalità di 5 * 4 = 20 Euro.

- **ESECUZIONE DEL SOFTWARE E STRUTTURA DEL FILE DI INPUT**
  Il software dovrà essere eseguito mediante la seguente linea di comando:

  ```
  pianificatore <percorso del file degli ordini>
  ```

  Ad esempio, se il comando dato fosse:

  ```
  pianificatore Ordini.txt
  ```

  il software caricherà gli ordini dal file Ordini.txt.
  Il file degli ordini dovrà avere un prodotto per riga, con tutti i parametri separati da virgola.

Ad esempio, se gli ordini fossero:

Identificativo: 4; Durata: 10; Scadenza: 12; Priorità: 4;
Identificativo: 12; Durata: 17; Scadenza: 32; Priorità: 5;

Il file dovrebbe contenere le seguenti righe:
```
4,10,12,4
12,7,32,1
```

Ogni file non può contenere più di 10 ordini.

- **MODALITA' DI UTILIZZO E DESCRIZIONE ALGORITMI**

  Una volta letto il file, il programma mostrerà il menu principale che chiede all'utente quale algoritmo di pianificazione dovrà usare. L'utente potrà scegliere tra i seguenti due algoritmi di pianificazione:

  > Earliest Deadline First (EDF): si pianificano per primi i prodotti la cui scadenza è più vicina, in caso di parità nella scadenza, si pianifica il prodotto con la priorità più alta.

  > Highest Priority First (HPF): si pianificano per primi i prodotti con priorità più alta, in caso di parità di priorità, si pianifica il prodotto con la scadenza più vicina.

  L'utente dovrà inserire il valore 1 per chiedere al software di utilizzare l'algoritmo EDF, ed il valore 2 per chiedere al software di utilizzare l'algoritmo HPF.
  Una volta pianificati i task, il software dovrà stampare a video:

  L'ordine dei prodotti, specificando per ciascun prodotto l'unità di tempo in cui è pianificato l'inizio della produzione del prodotto. Per ogni prodotto, dovrà essere stampata una riga con la seguente sintassi:

  ```
  ID:Inizio
  ```

  Dove ID è l'identificativo del prodotto, ed Inizio è l'unità di tempo in cui inizia la produzione.

  Inoltre verranno stampate anche l'unità di tempo in cui è prevista la conclusione della produzione dell'ultimo prodotto pianificato e la somma di tutte le penalità dovute a ritardi di produzione.

  ESEMPIO:
  ```
  Pianificazione EDF:
  4:0
  12:10
  Conclusione: 17
  Penalty: 0
  ```

  Una volta stampate a video le statistiche, il programma tornerà al menù iniziale in cui chiede all'utente se vuole pianificare la produzione utilizzando uno dei due algoritmi.
  L'uscita dal programma potrà essere gestita in due modi: si può scegliere di inserire una voce apposita (*esci*) nel menu principale, oppure affidarsi alla combinazione di tasti *ctrl-C*. In entrambi i casi però, tutti i file utilizzati dovranno risultare chiusi al termine del programma.

- **SALVATAGGIO OUTPUT SU FILE**
  Se l'utente inserisce due file come parametri da linea di comando, il file specificato come secondo parametro verrà utilizzato per salvare i risultati della pianificazione, indicando l'algoritmo usato. Ad esempio:

  ```
  pianificatore Ordini.txt Pianificazione.txt
  ```

  Il programma carica gli ordini dal file Ordini.txt e salva le statistiche stampate a video nel file Pianificazione.txt.
  Nel caso l'utente inserisca un solo parametro, la stampa su file sarà ignorata.

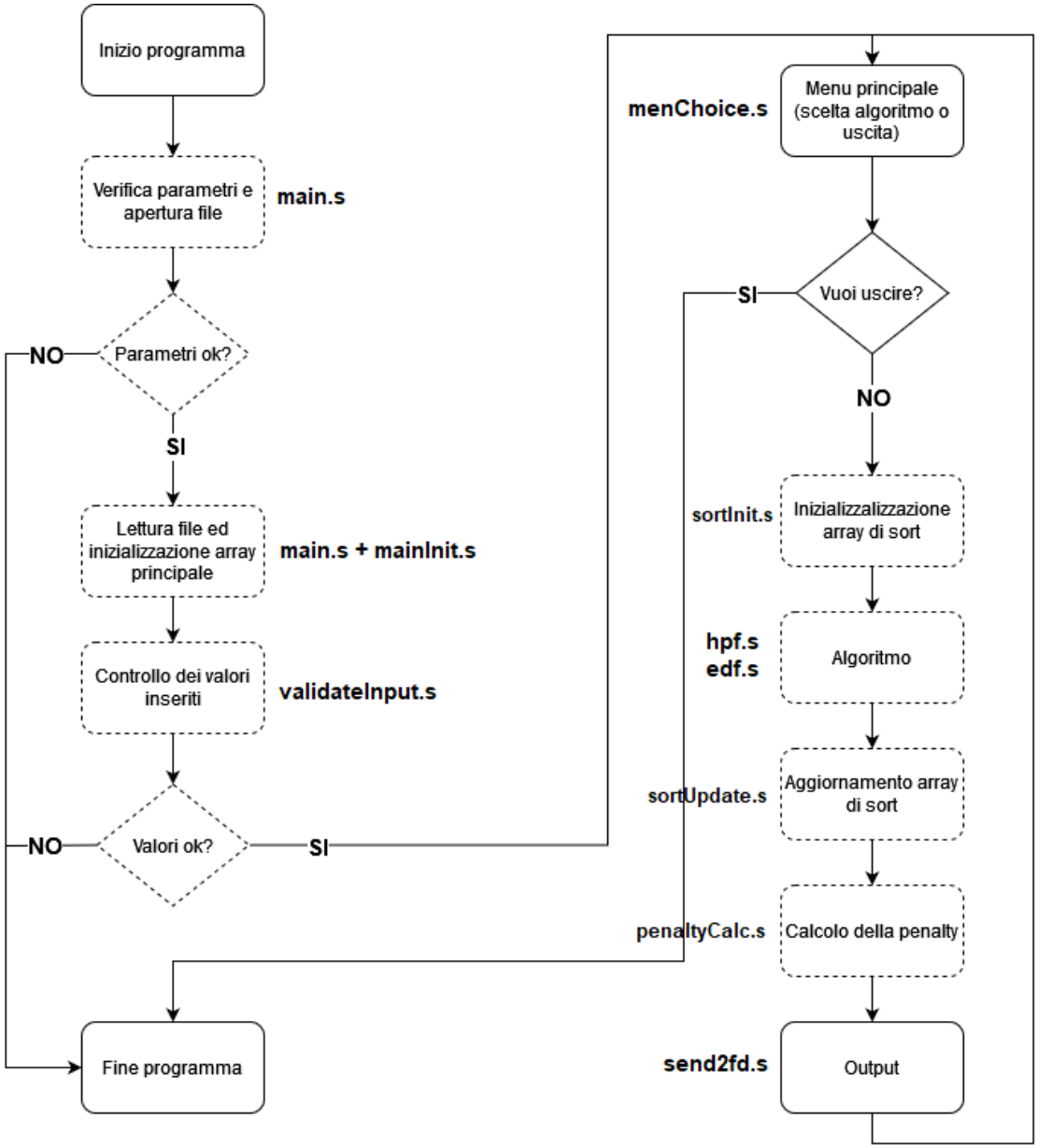
1.2. SPECIFICHE DI REALIZZAZIONE DEL SOFTWARE

Di seguito sono riportare le caratteristiche degli strumenti con cui e' stato realizzato il software:

Linguaggio	Assembly x86
Compiler	GAS (GNU assembler)
Linker	LD (GNU linker)
Flag di compilazione	--32 (compatibilita' con architetture a 32 bit)
Sintassi	AT&T
Kernel	Linux Kernel
Versione kernel	2.2

2. DESCRIZIONE FLUSSO DI ESECUZIONE DEL SOFTWARE

Verra' di seguito riprodotto un flowchart di alto livello che rappresenta il flusso di esecuzione del software considerando due diverse modalita': l'user experience e l'effective flow.



2.1. USER EXPERIENCE

Con il termine "User experiene" facciamo riferimento al flusso di esecuzione che l'utente sperimentera' durante l'esecuzione del programa.

Nello specifico, possiamo osservare che il flusso "user experience" e' rappresentato da tutti gli stati a linea continua cioe' gli stati visibili dall'utente.

L'interfaccia risulta quindi semplice e intuitiva. Cio' che si potra' osservare come utenti all'avvio del programma sara' un menu a tre opzioni: i due algoritmi e la voce *ESCI* per uscire dal programma. Una volta inserita la scelta il programma proseguira' flusso di esecuzione come indicato.

2.2. EFFECTIVE FLOW

L'*effective flow* si propone invece di rappresentare il flusso del programma da un punto di vista tecnico, considrando quindi anche i principali meccanismi implementati.

Questo flow, che coincide quasi del tutto con la struttura del file sorgente principale, e' rappresentato sia dagli stati a linea continua sia dagli stati tratteggiati, cioe' gli stati che devono essere "invisibili" durante il funzionamento del programma.

Sono rappresentati (sulla sinistra) le operazioni preliminari per poter accedere alla fase successiva del programma, e il ciclo (sulla destra) in cui: si sceglie la pianificazione, vengono eseguiti algoritmi di ordinamento, si calcola la penalty per poi terminare il ciclo con la fase di output dove i risultati sono stampati sul terminale ed eventualmente salvati nel file di output.

3. STRUTTURA DEL SOFTWARE IN FILE

Di seguito vengono elencati i file sorgenti che compongono il software, accompagnati da una breve spiegazione del loro funzionamento.
Successivamente si passera' ad un analisi piu' approfondita dei file principali.

3.1. ELENCO DEI FILE SORGENTI

NOME	DESCRIZIONE
main.s	File principale dove viene svolta la chiamata a tutte le funzioni. Nello stesso ci si occupa anche di verificare fin da subito i parametri passati dal main e si gestiscono eventuali errori di apertura del file.
mainInit.s	Procedura per inizializzare l'array dei prodotti con i valori contenuti nel file di input
validateInput.s	Procedura per controllare se i valori salvati nell'array dei prodotti rispettano i parametri
menChoice.s	Menu da cui selezionare un algoritmo di pianificazione o uscire dal programma
sortInit.s	Procedura per inizializzare l'array di sort con gli indirizzi di memoria del campo rispetto a cui ordinare i prodotti
edf.s	Algoritmo EDF che modifica gli indirizzi di memoria nell'array di sort
hpf.s	Algoritmo HPF che modifica gli indirizzi di memoria nell'array di sort
sortUpdate.s	Procedura per portare gli indirizzi di memoria dal campo puntato al campo ID
penaltyCalc.s	Procedura per calcolare la penalty complessiva
atoi.s	Procedura di conversione *array to integer* prendendo in input il primo valore sullo stack
itoa.s	Procedura di conversione *integer to array* utilizzando lo stack e prenendo in input EAX
send2fd.s	(*send output to file descriptor*) Procedura per stampare l'output nel file descriptor caricato

3.2. DESCRIZIONE E FUNZIONAMENTO DEI FILE PRINCIPALI

main.s:
Il main.s e' il file principale e si occupa della gestione del flusso del programma e di eventuali errori.
Le variabili utilizzate sono anche le uniche variabili che meritano di essere approfondite poiche' tutte le altre funzioni utilizzano o le stesse del main o flag e variabili temporali.

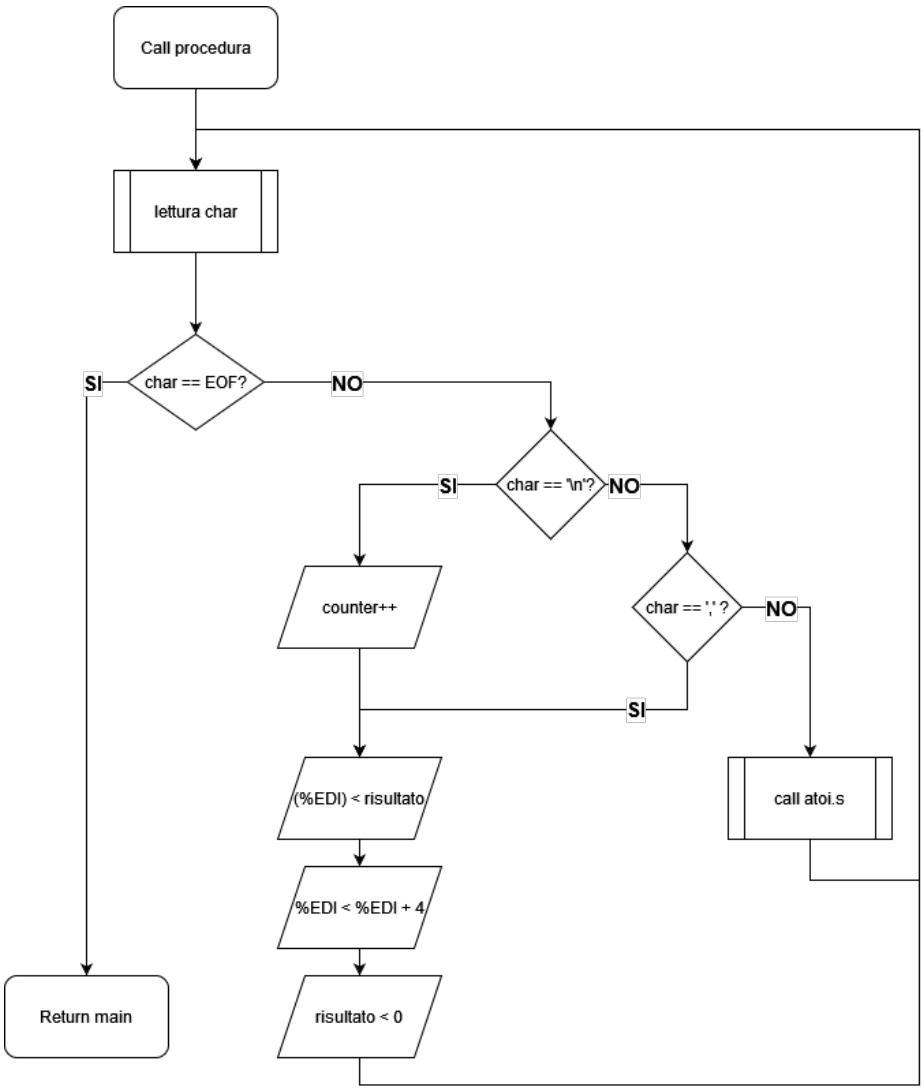
NOME VARIABILE	TYPE	SIZE	UTILIZZO
NO_NAME (file di input)	.ascii	variabile	Utilizzata per aprire il file di input. E' salvata nello stack fino al suo utilizzo
NO_NAME (file di output)	.ascii	variabile	Utilizzata per aprire il file di output. E' salvata nello stack fino al suo utilizzo (qualora ce ne fosse uno)
numero_parametri	.int	32 bit	Numero dei parametri passati dal main. Il suo impiego e' duplice: da una parte e' utilizzata per verificare che sia stato passato un numero adeguato di parametri e dall'altra e' impiegata per verificare se indirizzare o meno l'output al file di output
numero_prodotti	.int	32 bit	Indica il numero di prodotti che sono stati inseriti contandoli attraverso il carattere '\n'

algoritmo	.int	32 bit	Indica il numero dell'algoritmo scelto. Anche questa variabile viene utilizzata come flag all'interno di strutture condizionali
penalty	.int	32 bit	La penalita' complessiva calcolata a seconda dell'algoritmo
array_sort	.zero	32bit * x	Array per memorizzare tutti i valori del file (x = numero dei campo * numero dei prodotti)
array_prodotti	.zero	32bit * y	Array per memorizzare tutte le celle di memoria su cui performare lo swap (y = numero dei prodotti)

Il flusso di esecuzione del main.s e' rappresentato piuttosto accuratamente nella descrizione dell'*effective flow*.

mainInit.s
Riceve come parametri due valori del file register: il file descriptor del file di input salvato in EAX e l'indirizzo base dell'array dei prodotti salvato in EDI.
Le variabili locali utilizzate sono un contatore per salvare il numero di prodotti letti ed una variabile "risultato" per salvare il codice ASCII convertito in intero e memorizzarlo nell'indirizzo di destinazione.

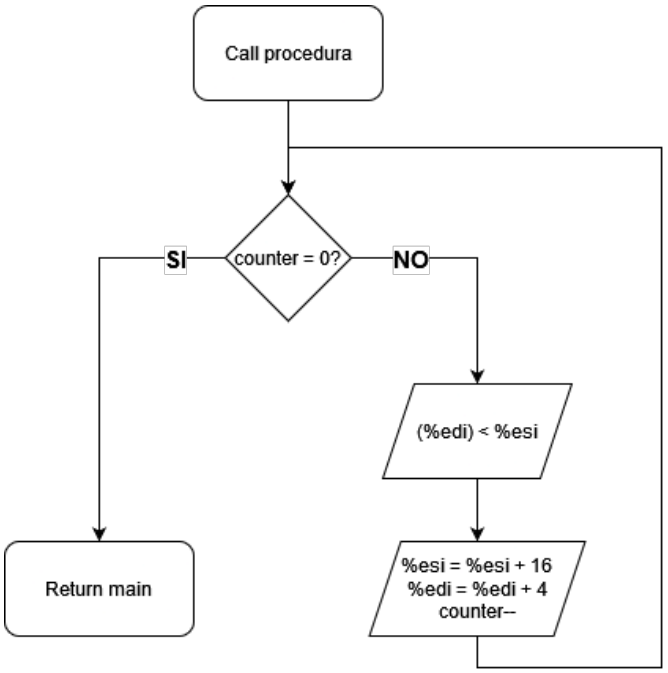
Prima della return il contatore viene spostato in EAX per esse poi salvato nel main.



sortInit.s
Questa funzione, analogamente alla precedente, inizializza un array ma questa volta l'array impiegato per il sort. Nello specifico vengono caricati nelle celle dell'array gli indirizzi di memoria dei parametri secondo cui ordinare i prodotti.
Se ad esempio si dovesse selezionare l'algoritmo HPF, nell'array di sort verebbero caricati tutti gli indirizzi delle priorita' dei prodotti. Al contrario, scegliendo l'EDF si sarebbero caricati gli indirizzi della scadenza.

I parametri passati sono il numero di prodotti che funge da counter per passare ogni prodotto in EAX, l'array dei prodotti in ESI e l'array di sort in EDI.

Di seguito sono riportati il flowchart e una rappresentazione grafica per semplificarne la compresione:

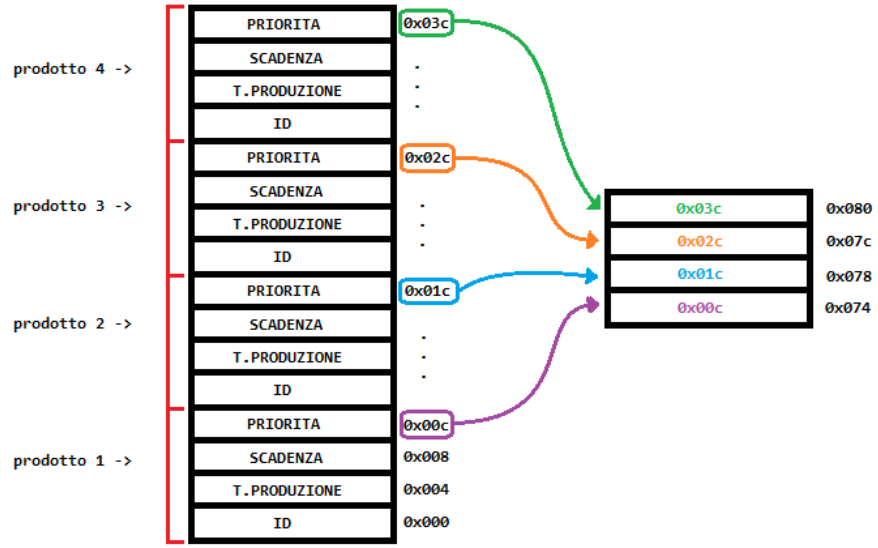


Il ciclo consiste nello scorrere il numero di prodotti e salvare ogni indirizzo nell'array per il sort.
Il proposito di utilizzare indirizzi di memoria e' quello di limitare il numero di swap da svolgere, riducendo di 4 volte tanto le operazioni richieste

Qui, possiamo apprezzare piu' chiaramente la riduzione delle dimensioni e quindi delle operazioni da svolgere.

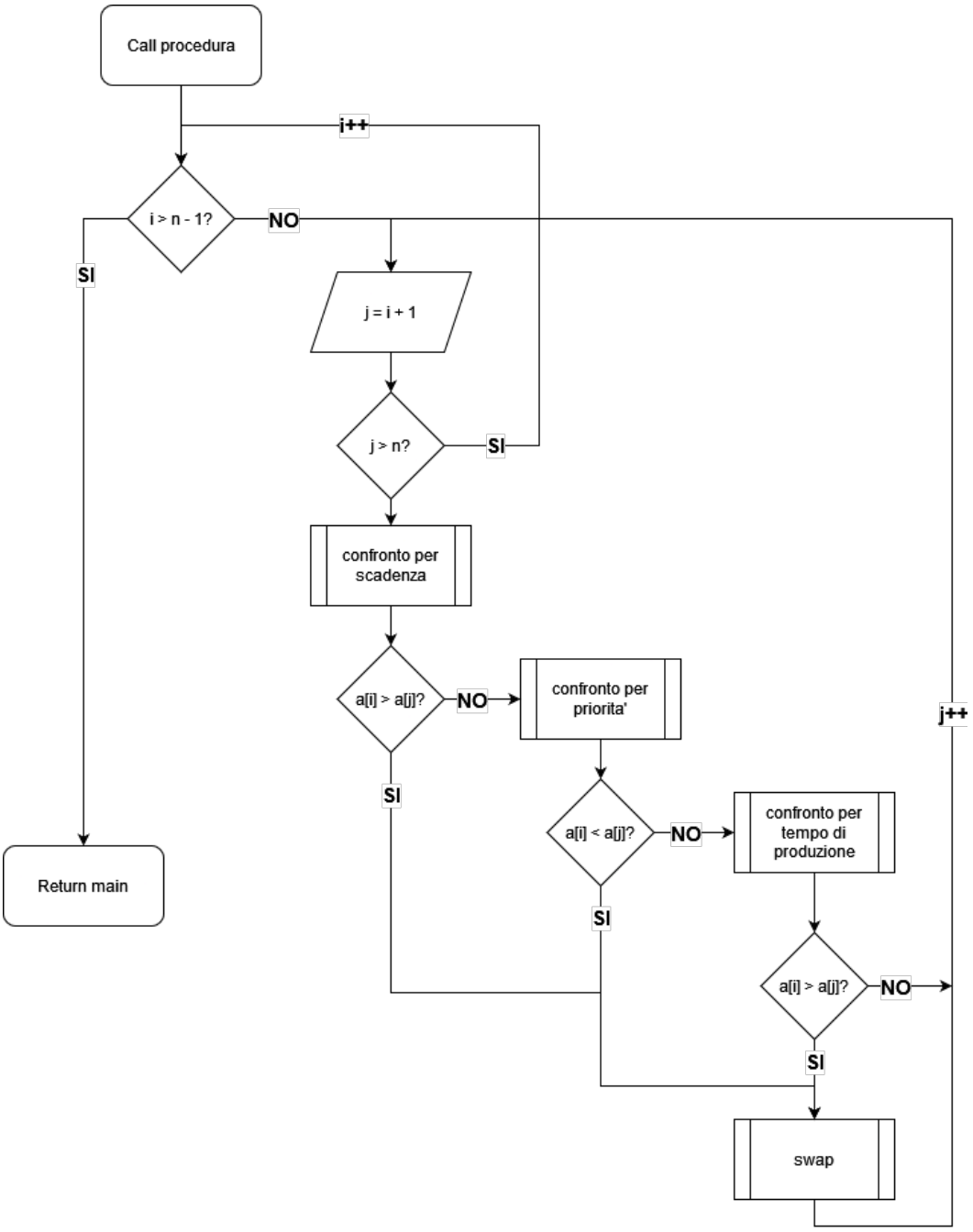
Inoltre, ogni swap consente di modificare "virtualmente" la posizione di ciascun elemento lasciando inalterato l'array principale.

Tuttavia, viene richiesta allocazione aggiuntiva di memoria che in certi casi potrebbe non essere impossibile o svantaggioso



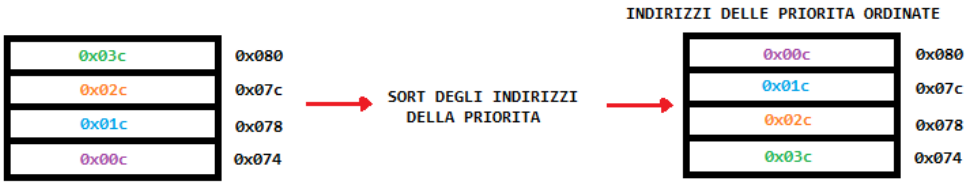
edf.s
Nell'algoritmo EDF, come in HPF, i parametri passati sono il numero di prodotti (come limite dei cicli) e l'indirizzo dell'array di sort su performare le operazioni di swap.

Di seguito verra' riportati il flowchart dell'algoritmo EDF. Segue di conseguenza HPF con leggere modifiche circa l'ordine dei campi da confrontare.

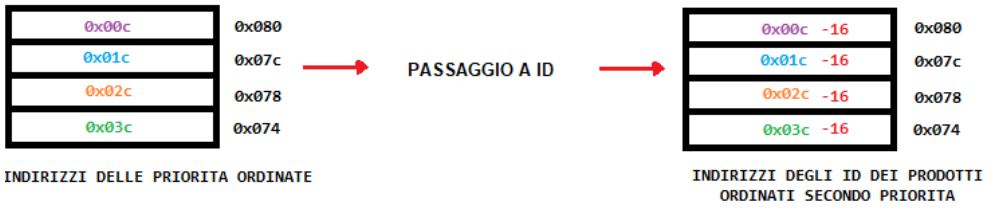


Nel flowchart l'array di sort viene indicato con la lettera "a" e gli offset usati per accedere ai campi sono rimpiazzati con gli indici "i" e "j" per maggior chiarezza.

Una volta terminato l'algoritmo di sort sull'array il risultato sara' il seguente (in questo caso si riprende l'esempio precedente con l'algoritmo HPF):



Successivamente all'algoritmo di ordinamento viene poi chiamata la procedura sortUpdate per riportare gli indirizzi salvati al campo ID:

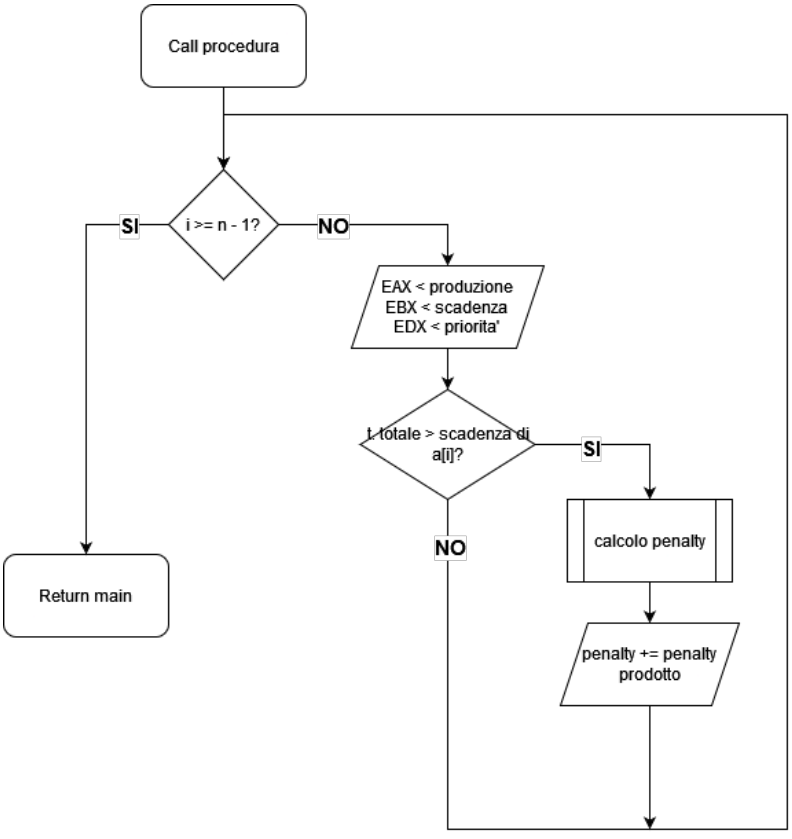


penaltyCalc.s

La seguente funzione si occupa di calcolare la penalty complessiva in base all'algoritmo selezionato in precedenza.

I parametri passati utilizzano sia lo stack, sia il register file.

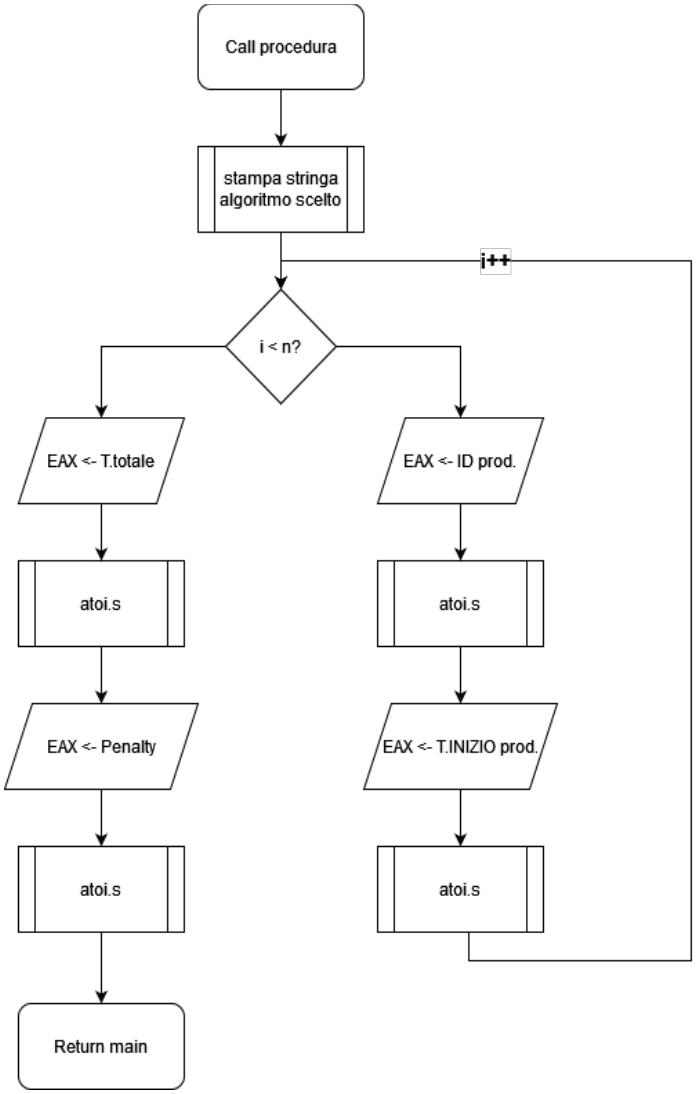
Vengono caricati il numero dei prodotti, che ancora una volta funge da counter per il ciclo in ECX, e l'indirizzo dell'array di sort in ESI, mentre, sullo stack, sono "pushati" il valore della variabile penalty (inizialmente a zero) e il valore l'immediato $0 che rappresenta il tempo complessivo.



send2fd.s
Questa procedura, situata al termine del ciclo, si occupa di stampare tutti i dati elaborati sino ad ora.
I parametri sono salvati principalmente sullo stack e sono: il il tipo di algoritmo scelto per poter selezionare le stringhe di output corrispondenti, il numero di prodotti (usato come limite per il ciclo), la penalty complessiva ed infine il file descriptor su cui si desidera stampare il risultato.
L'unico parametro passato su registro e' l'array di sort (dopo essere stato modificato dall'algoritmo di ordinamento) in ESI.

In primo luogo viene fatta una chiamata alla procedura per stampare i risultati a schermo (con file descriptor: 1), operazione che deve sempre essere eseguita.
Qualora il numero di argomenti fosse 3, e quindi ci fosse un file di output, la procedura verebbe nuovamente chiamata ma questa volta specificando il file descriptor assegnato al file di output.

Flowchart:



La procedura atoi.s sfrutta lo stack per scomporre un numero, che deve essere caricato su EAX.
Inizialmente vengono pushate le cifre dalla meno significativa alla piu' significativa sfruttando il resto della divisone che si trovera' in EDX.

Una volta terminata la scomposizione, con l'operazione inversa, la pop, vengono stampate le cifre pushate una ad una, sfruttando la struttura LIFO dello stack. Si ricompone cosi' il numero sottoforma di caratteri ASCII.

4. TESTING

4.1. File di input – gestione eccezioni ed errori

Di seguito riportati tutti gli errori gestiti dal software

Input	Errore/Warning	Esecuzione
File vuoto	Errore file vuoto	**TERMINATA**
File senza ','	Errore parametri non conformi	**TERMINATA**
File senza '\n' tra prodotti	Warning un prodotto + Errore prametri non conformi	**TERMINATA**
File con 1 prodotto valido	Warning un prodotto	**ALTERATA**
File con 1 prodotto non valido	Warning un prodotto + Errore prametri non conformi	**TERMINATA**
File con 2+ prodotti di cui almeno 1 non valido	Errore parametri non conformi	**TERMINATA**
File con 11+ prodotti	Errore numero prodotti superiore al limite	**TERMINATA**
File con caratteri diversi da 0-9, ',' , '\n'	Errore parametri non conformi	**TERMINATA**

4.2 Input da tastiera – gestione eccezioni ed errori

L'inserimento da tastiera e' gestito tramite un loop che non termina fino a che non e' stato inserito un input valido cosi' da impedire comportamenti inaspettati o uscite improvvise.

E' stata impiegata una stringa .ascii di 10 caratteri per impedire un overflow e di conseguenza comportamenti imprevedibili nel caso si dovessero inserire molti piu' caratteri di quelli necessari.

4.3 Parametri – gestione errori

Qualora si dovesse verificare un errore di apertura del file di input o di quello di output il processo avverte l'utente con un messaggio di errore apposito ed termina la sua esecuzione.

5. SCELTE PROGETTUALI

- Abbiamo scelto di suddividere il progetto in più file in modo da avere maggiore chiarezza in fase di sviluppo, evitando così di commettere errori banali legati a una difficile comprensione del codice.

- Abbiamo deciso di utilizzare un array per i campi dei prodotti viste le ridotte dimensioni del file di input e per maggior semplicita' e sicurezza durante il processo di manipolazione e accesso ai dati.

  In effetti, questa scelta comporta un utilizzo maggiore della memoria ed un possibile spreco nel caso in cui i prodotti letti non siano molti.
  In architetture embedded l'utilizzo eccessivo di memoria potrebbe portare ad un aumento considerevole dei costi di produzione, spece se i chip dovessero essere realizzati per un mercato su larga scala. Sarebbe quindi piu' conveniente ottimizzare per l'area e sfruttare la frequenza dei processori moderni per ottenere un risultato ottimale.

  Tuttavia, data la natura del progetto, il software sembrerebbe avere sembianze piu' simili a quelle di un tool utilizzabile su personal computer di un'azienda per ottenere statistiche utili all'organizzazione della produzione.
  Se l'assunzione dovesse essere vera, a questo punto il problema rimarrebbe limitato alle dimensioni del file.

- Abbiamo scelto di utilizzare un secondo array per le stesse motivazioni fornite nel punto precedente.
  In particolare, abbiamo osservato che, con un utilizzo ancora piu' ridotto di memoria, sarebbe stato possibile realizzare un array di puntatori su cui performare gli algoritmi di ordinamento in modo da operare lo swap delle posizioni in 4 passaggi al posto che 12 (rapporto 1:3).

- Abbiamo deciso di gestire le stampe all'interno dei file in cui si sarebbe dovuta performare la stampa, senza creare funzioni apposite, data la ridotta quantita' di stringhe da visualizzare.
  Lo stesso vale per la gestione dei messaggi di errore/warning che sono interamente gestiti nel main in una sezione del codice dedicata.
  Si sarebbe potuto sicuramente dividere ulteriormente il codice in sezioni interamente dedicate alla stampa ma abbiamo appurato che il codice scritto e' sufficentemente chiaro e non c'e' bisogno di ulteriori suddivisioni.

- Per quanto la procedura per stampare/salvare gli output sarebbe potuta essere stata sviluppata in modo da venir chiamata una sola volta, abbiamo preferito rendere questa funzione piu' flessibile aggiungendo ai parametri anche il file descriptor di destinazione a cui si vuole mandare l'output.